

1993-09

# Improvements to autonomous forces through the use of genetic algorithms and rule base enhancement

Jacobs, Robert Alan.

Monterey, California. Naval Postgraduate School

# NAVAL POSTGRADUATE SCHOOL
## Monterey, California




# THESIS

IMPROVEMENTS TO AUTONOMOUS FORCES
THROUGH THE USE OF GENETIC ALGORITHMS
AND RULE BASE ENHANCEMENT

by

John Phillip Steiner
and
Robert Alan Jacobs

September 1993

Thesis Advisor: Hemant K. Bhargava

Approved for public release; distribution is unlimited

Unclassified
Security Classification of this page

## REPORT DOCUMENTATION PAGE

| | | | |
|---|---|---|---|
| 1a Report Security Classification: Unclassified | | 1b Restrictive Markings | |
| 2a Security Classification Authority | | 3 Distribution/Availability of Report: Approved for public release; distribution is unlimited | |
| 2b Declassification/Downgrading Schedule | | | |
| 4 Performing Organization Report Number(s) | | 5 Monitoring Organization Report Number(s) | |
| 6a Name of Performing Organization: Naval Postgraduate School | 6b Office Symbol (if applicable) 37 | 7a Name of Monitoring Organization: Naval Postgraduate School | |
| 6c Address (city, state, and ZIP code): Monterey CA 93943-5000 | | 7b Address (city, state, and ZIP code): Monterey CA 93943-5000 | |
| 8a Name of Funding/Sponsoring Organization: ARPA/ASTO | 6b Office Symbol (if applicable) | 9 Procurement Instrument Identification Number | |
| Address (city, state, and ZIP code) | | 10 Source of Funding Numbers | |
| | | Program Element No | Project No / Task No / Work Unit Accession No |

11 Title (include security classification) Improvements to Autonomous Forces Through the Use of Genetic Algorithms and Rule Base Enhancement

12 Personal Author(s) John Phillip Steiner and Robert Alan Jacobs

| 13a Type of Report | 13b Time Covered From To | 14 Date of Report (year, month, day) | 15 Page Count |
|---|---|---|---|
| Master's Thesis | | 1993 September | 94 |

16 Supplementary Notation The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

| 17 Cosati Codes | | | 18 Subject Terms (continue on reverse if necessary and identify by block number) |
|---|---|---|---|
| Field | Group | Subgroup | Genetic Algorithm, Autonomous Forces, NPSNET, Expert System, Rule Base Enhancement |
| | | | |

19 Abstract (continue on reverse if necessary and identify by block number)

This thesis discusses two approaches to enhancing the performance of intelligent autonomous agents in a computer combat simulation environment so that their performances more closely model the tactical decisions made by human players. The first approach addresses incorporating a genetic algorithm (GA) into the NPSNET Autonomous Force Expert System (NPSNET AF), while the second approach focuses on enriching the existing rule base and decision strategies. First, we develop a functional genetic algorithm with the intent of providing dynamic, real-time learning within the NPSNET AF. However, we conclude that the GA is better suited for a static problem, such as artillery battery registering of fires, rather than for the dynamic battlefield of the NPSNET. Second, we enrich the NPSNET AF expert system by enabling it to choose from among four formations and by providing a mechanism for transitioning between them. We enable the expert system to make formation decisions based upon general terrain characteristics and target location.

| 20 Distribution/Availability of Abstract | 21 Abstract Security Classification | |
|---|---|---|
| XX unclassified/unlimited __ same as report __ DTIC users | Unclassified | |
| 22a Name of Responsible Individual: Hemant K. Bhargava | 22b Telephone (include Area Code): (408) 656-2264 | 22c Office Symbol: AS/BH |

DD FORM 1473,84 MAR — 83 APR edition may be used until exhausted; All other editions are obsolete — security classification of this page: Unclassified

Approved for public release; distribution is unlimited

Improvements to Autonomous Forces
Through the Use of Genetic Algorithms and Rule Base Enhancement

by

Robert Alan Jacobs
Captain, United States Marine Corps
B.A., University of Mississippi, 1983

and

John Phillip Steiner
Lieutenant, United States Navy
B.S., University of New Mexico, 1986

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN INFORMATION TECHNOLOGY MANAGEMENT

from the

NAVAL POSTGRADUATE SCHOOL

Authors: [redacted]

Robert Alan Jacobs

[redacted]

John Phillip Steiner

Approved By: [redacted]

Prof. Hemant K. Bhargava, Thesis Advisor

[redacted]

Prof. Balasubramaniam Ramesh, Associate Advisor

[redacted]

Dr. David Pratt, Associate Advisor

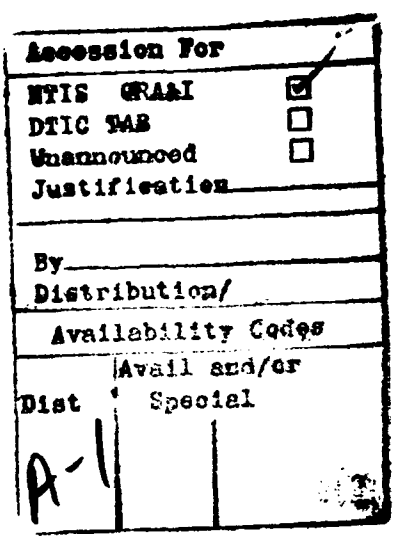
[redacted]

Prof. David R. Whipple, Chairman, Department of Administrative Sciences

# ABSTRACT

This thesis discusses two approaches to enhancing the performance of intelligent autonomous agents in a computer combat simulation environment so that their performances more closely model the tactical decisions made by human players. The first approach addresses incorporating a genetic algorithm (GA) into the NPSNET Autonomous Force Expert System (NPSNET AF), while the second approach focuses on enriching the existing rule base and decision strategies. First, we develop a functional genetic algorithm with the intent of providing dynamic, real-time learning within the NPSNET AF. However, we conclude that the GA is better suited for a static problem, such as artillery battery registering of fires, rather than for the dynamic battlefield of the NPSNET. Second, we enrich the NPSNET AF expert system by enabling it to choose from among four formations and by providing a mechanism for transitioning between them. We enable the expert system to make formation decisions based upon general terrain characteristics and target location.





## TABLE OF CONTENTS


I. INTRODUCTION ..... 1

A. Objectives ..... 1

B. Motivation ..... 2

1. Intelligent Agents ..... 2

2. Mature Rule Base ..... 3

C. Proposed Solution ..... 3

1. Apply Genetic Algorithm (GA) As A Machine Learning Tool ..... 3

2. Add Formation Control Rules to the Rule Base ..... 4

D. Thesis Organization ..... 4

II. BACKGROUND ..... 6

A. Simulation ..... 6

1. Importance of Simulation ..... 6

2. Types of Autonomous Simulation ..... 7

B. Background of NPSNET AF ..... 8

C. Natural Evolution ..... 9

D. Genetic Algorithms ..... 11

E. Expert Systems And Machine Learning ..... 12

III. MACHINE LEARNING FOR THE AUTONOMOUS FORCE ..... 14

A. Static And Dynamic Systems ..... 14

B. Real-Time And Historical Learning ..... 16

IV. ANALYSIS OF AUTONOMOUS FORCE MOVEMENT ..... 18

A. Single-Agent Movement ..... 18

B. Multiple Agent Movement ..... 19
C. Selecting The Best Formation ..... 19
1. Movement of Agents in Response to Environmental Stimuli ..... 19
2. Sufficient Sampling ..... 20
3. Mission Objectives ..... 22
4. Enemy Agents ..... 22
5. Selecting The Best Formation by Applying a Genetic Algorithm ..... 22
D. Movement Through A Turn ..... 23
E. Keeping Agents in Formation ..... 25
F. Formation Selection ..... 25
G. The Echelon n Formation Transition ..... 27
H. Realistic Transition ..... 29
I. Constraining The Turn ..... 32
V. APPLICATION OF GENETIC ALGORITHMS TO AUTONOMOUS FORCES ..... 34
A. The Genetic Algorithm ..... 34
1. Definition Of A Genetic Algorithm ..... 34
2. Reasons For Using A Genetic Algorithm ..... 34
3. The Mechanics Of A Genetic Algorithm ..... 35
4. Details of the Functioning GA ..... 36
5. Tailoring the GA ..... 39
B. Benefits ..... 41
C. Limitations ..... 42

D. Approaches ..... 43
1. Our Approach ..... 43
2. Reason For Delaying Integration Of The GA With NPSNET ..... 43
VI. CONCLUSION ..... 47
APPENDIX A ..... 50
APPENDIX B ..... 74
LIST OF REFERENCES ..... 80
INITIAL DISTRIBUTION LIST ..... 84

## ACKNOWLEDGEMENT

We would like to express our sincere appreciation for the funding support provided by LtCol. David Neyland, USAF, of ARPA/ASTO. We would also like to thank Drs. Hemant Bhargava, B. Ramesh, David Pratt, and Michael J. Zyda of the Naval Postgraduate School for their tremendous guidance, insight, and intellectual support. Most of all, we would like to thank our wives, Darla and Kathy, for picking up the slack in so many ways during untold hours of our absences while we worked on this project.

# I. INTRODUCTION

The requirements of modern warfare demand that battlefield commanders be capable of making increasingly complex decisions. In order to make these decisions, these commanders need extensive training support. The real-world training required by the commanders can be extremely expensive and dangerous. Computer simulation in which players compete against autonomous agents can provide a cost-effective and less hazardous training medium. In order for this training to be of value, though, autonomous agents must have tactical decision-making capabilities similar to the human players.

## A. OBJECTIVES

The goal of our work is to enhance the performance of intelligent autonomous agents in a combat simulation environment so that their performances more closely model the tactical decisions made by human players. We focus our efforts on the NPSNET autonomous forces system (NPSNET AF) (Bhargava and Culpepper, 1992). Specifically, we attempt to improve the human-like qualities of the NPSNET AF by providing it with the capability to learn from its actions through the application of a genetic algorithm (GA) (Goldberg, 1989) and by improving its existing rule base in three ways. First, we provide agents with more formation choices. Second, we provide the agents with the ability to transition between formations. Last, we provide the agents with a decision-making strategy for selecting a particular formation.

## B. MOTIVATION

### 1. Intelligent Agents

Combat simulation with autonomous forces can be a cost-effective and safe training aid. Full scale battlefield maneuvers require a dedicated military force complete with combat equipment. Such maneuvers are generally very costly. Computer simulation provides a possible avenue for reducing these costs. Computer simulation also provides a safe learning environment in which the participants can train and make mistakes without the serious health risks associated with live weapons fire and personnel and machine maneuvers (Braudaway, 1993). An autonomous system requires fewer users to operate it than either a semi-autonomous system or a non-automated system. With an autonomous system, the user can simulate training scenarios in which the user competes with a hostile force without the need for extra personnel to assume the role of the opposing force.

In order for an autonomous system to be credible, though, it must depict computer-generated agents in a way humans consider realistic (SikSik, 1993). Though rule-based systems provide a means of emulating human decision making, they generally do not learn from their actions. In other words, if a non-stochastic, rule-based system is given the same set of stimuli over and over, it repeats the exact same actions, no matter what the consequences of those actions are. People, on the other hand, learn from their actions. The results of the action provide feedback to the person who initiates the action. If the results are favorable, the person tends to continue with the same action given the same response. If the results are only partially favorable, the person attempts to improve his performance by altering the

actions he/she undertakes given the particular set of stimuli. If the results are unfavorable, the person generally tries something completely different. For an autonomous force system to be truly credible and of a realistic training value, it should be capable of learning from its actions. An autonomous force system based solely upon the use of an expert system is not ideal. The system should also include a machine learning methodology which gives the system the ability to learn new actions based upon feedback it receives from its previous actions.

### 2. Mature Rule Base

Combat simulation requires a realistic environment in which agents within the environment display plausible behavior. Agent behavior within the NPSNET AF is maintained within its rule base. Therefore, a mature rule base that captures realistic behavior and capabilities is essential. One of our goals is to provide the system with the ability to perform more realistically, through an expanded or "matured" set of rules.

## C. PROPOSED SOLUTION

### 1. Apply Genetic Algorithm (GA) As A Machine Learning Tool

For the NPSNET AF to be a worthwhile training environment, it should be capable of learning through some machine learning technique. Without learning, the NPSNET AF does not emulate human behavior to its fullest potential because it currently does not learn from its past actions. We address the use of a GA as a machine learning tool for the NPSNET AF. We do so because the GA is a machine learning technique in which the system may learn new actions based upon feedback from previous actions. We initially chose to use a GA to provide real-time learning in a dynamic environment.

However, we conclude that this approach is incorrect. We believe that providing the NPSNET AF with the ability to emulate battlefield learning through a genetic algorithm learning system such as the *anytime learning* system proposed by Grefenstette and Ramsey (1992) will eventually enhance the realism of the NPSNET AF training environment.

**2. Add Formation Control Rules to the Rule Base**

Not only must the NPSNET AF be capable of making intelligent decisions, but it must also be capable of performing believable actions based on its decision rules. Therefore, we enhance the NPSNET AF rule base in a three step approach. We make these enhancements in the area of platoon/tank movement. First we give the AF the power to choose between four formations based upon target location and general terrain characteristics. Second we provide a mechanism, which we call the echelon n, for transitioning between formations. Last, we give the system the ability to properly maneuver the AF in the selected formation.

## D. THESIS ORGANIZATION

In Chapter II, we discuss the importance of computer simulation and provide background information on the NPSNET autonomous forces, on natural evolution, on genetic algorithms, and on expert systems and machine learning. In Chapter III, we discuss static vs. dynamic systems and the applicability of dynamic, real-time learning capabilities to the NPSNET AF. In Chapter IV, we analyze autonomous force movement and describe changes we make to the NPSNET AF, including our echelon (n) formation transition algorithm. In Chapter V we discuss how genetic algorithms work and our reasons for not implementing a genetic algorithm within the NPSNET AF.

Lastly, in Chapter VI, we summarize our work and address possible future efforts for enhancing the NPSNET AF.

# II. BACKGROUND

## A. SIMULATION

### 1. Importance of Simulation

One of the greatest values of automation lies in using a computer to study the performance of real-world systems without the expense or hazards normally associated with these systems. Computer simulations can help us predict and understand events in the real world if the model is sufficiently accurate and believable. Of five basic reasons for using automated simulation systems (Reddy, 1993) -- optimization, "what-if" scenarios, proof-of-capability, requirements analysis, and personnel training -- we address the last in this thesis. Anderson et al. (1991), believe that computer simulations can, in fact, produce a more efficient and/or effective model of the real-world system.

Although all training can be computer-simulated in some manner or another, we believe that not all training should be. The spectrum of practicality in computer simulation, specifically, runs from the trivial to the nearly critical. At one end of the continuum, it is hard to imagine preparing troops for combat by using a computer to simulate running or lifting weights. However, at the other end of the spectrum, some tasks practically mandate the need for simulation. A good example is that of flight training simulators, which are used to prevent loss of life and aircraft due to the inexperience of student pilots.

Personnel training simulators run from systems for training individuals to systems for conducting staff-level training. Although the benefits from

simulation are often hard to quantify, few would disagree as to the need for improving combat simulators. Our work focuses on making the simulation of the NPSNET AF more realistic by attempting to give the user a more intelligent and believable opponent force against which to fight.

## 2. Types of Autonomous Simulation

A simulation which uses autonomous forces can call for a variety of methods in which a user may interact with the system and play against an AF. The least interactive method is a fully autonomous simulation in which there is no human interaction. The user generally watches a scenario but provides no controlling input. This requires a scripted battle, perhaps based on actual past battles, to be programmed into the autonomous behavior of the system, and it is generally meant to teach a specific lesson to the user without his input. These types of simulators can be found in state parks and museums. They serve a military purpose by providing a means to analyze and learn from previous battles.

The second, or semi-autonomous (Gat, 1993) method of interaction lets the user control some elements. This method calls for a "man-in-the-loop" and and the agents in this environment are called semi-autonomous forces (SAF). At this level, the user can direct the actions of agents, or he can relinquish control to the computer, at which time the vacated agent becomes autonomous.

NPSNET AF allows both autonomous and semi-autonomous scenarios. The user may either control his own vehicle or simply ride in an autonomous vehicle which he may not control. The NPSNET AF forces are autonomous agents over which a human player has no control.

## B. BACKGROUND OF NPSNET AF

The NPSNET AF is the expert system which controls a computer-generated autonomous force (AF). NPSNET AF was developed by Captain Michael E. Culpepper, USA, (Culpepper, 1992), and Captain William C. Branley, USA, (Branley, 1992) to further the goal of producing a viable personnel training system. This autonomous force runs on Silicon Graphics workstations in the NPS Computer Science laboratory. NPSNET AF directs the movement of autonomous agents throughout the NPSNET battlefield with no human interaction required for controlling the movement of the AF.

NPSNET is a three-dimensional, virtual-world, combat modeling system. It uses a backbone ethernet system which the autonomous force can tap into. NPSNET provides a 3-D graphical environment representing the training area at Fort Hunter-Liggett, CA. It uses a flat-world model which does not simulate earth curvature (Pratt, 1992). The NPSNET code is responsible for tracking and displaying all vehicles a user sees at his terminal, whether they be generated by the NPSNET AF, by other player programs, or by a local SIMNET database.

The NPSNET AF is one of many independent processes (or player programs) running on a network that communicates with the NPSNET. Communication between the NPSNET AF and the NPSNET is done by sending and receiving vehicle state messages via a communications routine known as the network demon (Pratt, 1992).

Culpepper developed a tactical decision-making module, which is used in the NPSNET AF. The principles and heuristics used by the module are implemented in CLIPS, and the module is fully integrated with NPSNET. It

tells NPSNET where the AF vehicles are, where they are going, and what the status of each vehicle is. The tactical decision-making module is divided into three sub-modules. They are 1) tactical command and control, 2) movement and route planning, and 3) target engagement. The command and control sub-module generates mission-oriented goals for the other two sub-modules (movement and route planning, and target engagement).

Once the command and control sub-module generates a movement objective, the movement and route planning sub-module determines the action necessary to move the vehicle towards the objective in the following three stages: establish a march route to the objective, make platoon movement decisions, and make decisions as to individual tank movements (Culpepper, 1992).

Battlefield decisions are made at three basic levels: individual, crew, and unit. These are distinct decisions on the battlefield, and they are distinct within the tactical decision making module of the computer simulation environment. Generally, as one progresses from individual to unit, the level of decision making becomes more complex. Issues of what must be done supersede concerns for how things must be done (Culpepper, 1992). We feel this higher level of decision-making should be the focus of improvements to the NPSNET AF. Accordingly, our rule-base-enhancement work focuses on the tactical command and control sub-module.

## C. NATURAL EVOLUTION

English Evolutionist Charles L. Darwin (1809-1882) applied evolution theory to the natural world. Plants and animals, he theorized, are in a constant struggle for food, water, space, and protection against predators and

the environment itself. In his first book (Darwin, 1859), he fortified the then-current theory on evolution and natural selection, espousing two main premises: 1) some species stay in their environment and crowd others out, and 2) other species will find new environments in which they can flourish. From Darwin's work, it follows that when the environment changes significantly, an organism must do one of three things: 1) adapt to survive in the new environment, 2) move to a new environment where conditions are relatively similar to the old one prior to the change, or 3) perish.

We note numerous similarities between real combat and natural evolution. In both of these cases, the fittest individuals survive, while lesser ones either move to a favorable environment or perish. We note that since a genetic algorithm is based on the principles of natural evolution, which encompass genetic "learning" from generation to generation, a GA seems to be a good candidate to provide machine learning to the NPSNET AF battlefield simulator.

Genetic learning depends, in large part, on the notion of fitness. Fitness is the measure of an individual's likelihood of survival in a given environment. On fitness, John Holland (1992) states:

> Roughly, the fitness of a phenotype [individual] is the number of its offspring which survive to reproduce in the next generation... The fitness of an individual is clearly related to its influence upon the future development of the population.

The purpose of the genetic algorithm is to constantly evolve toward the best or fittest problem solution, moving from one population to another, and remembering the best of each population as it goes. In the GA, one set of

input parameters (one possible solution set) to a problem under study is represented as an individual. The individual's fitness is the suitability of these inputs to solve the problem.

Darwin proposed that competition is an inherent part of survival of a species. Following this competition theory, the genetic algorithm arrives at its recommended solutions to a problem -- the strongest solutions survive and are "memorized" and the weak solutions "perish"-- as the GA migrates toward its best recommendation.

## D. GENETIC ALGORITHMS

The "father of genetic algorithms," John Holland (1992), describes a genetic algorithm as a programming technique for solving complex problems which most humans cannot even begin to understand. The GA manipulates strings of input parameters, which are encodings of solution sets to a problem, and its goal is to find the best probable solution to the problem in question, given a certain amount of time. While classical search techniques start with an initial solution set and perform exhaustive searches for all possible solutions, the GA conducts a guided, stochastic search through the solution space.

A classical problem which can be solved by a search algorithm such as the GA is the fifty-city Traveling Salesman Problem (TSP). This problem involves a salesman who must visit each of fifty cities, and he wishes to travel along the shortest possible path. According to Goldberg, "...[the TSP] is a member of a class of problems believed to be unsolvable in polynomial time." (Goldberg, 1989). The brute-force, exhaustive search method of finding the best answer to this problem is to search every possible path. Since there are

roughly 50 factorial, or $3.04 \times 10^{64}$, ways for the salesman to complete his task, some method of guiding the search is necessary. The average human can find a solution to this problem intuitively by simply connecting the city points on a map and measuring the distance along his chosen path. He does so by informally guiding his own search. He makes his best guess by discounting seemingly unreasonable solutions and focusing on what he believes to be the most likely route. Though he has chosen a path, he can never be sure that it is the best one.

Like the human, a genetic algorithm also conducts a guided search for an optimal solution. A genetic algorithm uses an iterative search process. During the first iteration, it initially selects a number of random solutions from the search space and evaluates their ability to meet the shortest-distance criteria of the TSP. On each iteration thereafter, the GA guides its search by emphasizing the better probable solutions and deemphasizing the lesser ones. The GA continues to search until it has met some pre-defined search criteria, or until time expires on the search. At this point the GA arrives at its best probable solution. However, one cannot say for certain that this solution is the absolute optimal solution to the problem. It is what the GA believes to be the most-likely optimal solution. Though there is no guarantee that the GA will find the optimal solution to a problem, the GA will most likely find solutions which are approximately optimal or "good enough" (Michalewicz, 1992).

## E. EXPERT SYSTEMS AND MACHINE LEARNING

The NPSNET AF system is an expert system. It manipulates rules and considers facts to move agents on the battlefield. To understand the

NPSNET AF system, it is useful to know something of the background of expert systems in general. In the 1970's, researchers focused their efforts on modeling the knowledge of "experts." Knowledge engineering, as it has come to be known, has its roots in the belief that human expertise can be captured in computer programs. A human expert possesses vast knowledge and uses intuitive reasoning to arrive at quick solutions based on only relevant facts. Expert systems reach a decision based on a focused and probabilistic search of known facts and rules. The expert system selects only the pertinent rules for a given problem (focusing) and reaches the solution most likely to be the "best" (probabilistic solution-finding) (Forsyth, 1989). This focusing ability is an important strength of expert systems and is vital to computer simulations, because the less time a system spends in making decisions, the more computational effort it can devote to making graphic representations of a simulated battle more realistic, and realism is an important aspect of graphical computer simulation.

Machine leaning is the concept of developing computer systems that simulate intelligence (Winston, 1984). The goal is not to replace humans with computers, but merely to make computers that are capable of learning new behaviors based upon certain input stimuli. One aspect of machine learning, called cognitive computing, denotes a set of problem solving methodologies which mimic the learning processes found in nature (Johnson, 1993). One such cognitive computing technique is the genetic algorithm. We look at GAs because we view battlefield learning as an evolutionary process, and the GA uses the precepts of natural evolution in conducting its search for an optimal problem solution.

# III. MACHINE LEARNING FOR THE AUTONOMOUS FORCE

The NPSNET AF operates in a dynamic environment. NPSNET AF agents operating within the environment elicit stimulus-response behavior as defined by the system rule-base. Though the agents respond to their environment, they do not learn new behaviors in response to a changing environment or as a result of previous undersirable performance. We believe that in order for autonomous agents to provide realistic responses in a computer combat simulation environment, the agents need to be able to learn how their actions affect their environment and how to tailor their actions in response to a dynamic environment. There is not, however, only one method of building a machine learning element into a computer system. The methodology depends on the environment in which the machine will operate. Next we discuss the concepts of static and dynamic systems as well as real-time and historical learning. These concepts are important considerations when devising a learning system.

## A. STATIC AND DYNAMIC SYSTEMS

Static systems generally use a pre-defined, scripted approach to solving a problem. Agents within a static system usually perform projective planning, and such planning involves global evaluation of the environment. Dynamic systems, however, are not scripted. They are locally oriented and utilize reactive planning, responding to local stimuli (Schultz, 1991). Both types of systems have benefits and weaknesses.

The predefined nature of a static system is both a strength and a weakness of the system. Predictability and/or reliability is a strength. These systems

generally perform a specified task, or when given a particular input, they produce a specific output. Such behavior is beneficial for conducting repetitive, well-understood tasks. Static systems may also be used for training people to respond correctly in a given situation or for testing required specifications in a design. However, battle situations, including those encountered by the NPSNET AF within the NPSNET, are often ambiguous and require a degree of flexibility which static systems usually do not provide.

The pre-defined nature of a static system also limits the system in another way. By its very nature, a static system does does not rely on feedback from the environment for tailoring its actions. It does not care about the consequences of its actions. The actions for the system are always the same for a particular input. The NPSNET AF continually interacts with its environment and needs to be aware of input/output relationships (Henderson, 1991).

The finite nature of a static system enables the system to look at its global state and analyze all available inputs. Such behavior, though beneficial in solving problems, often requires significant CPU resources and detracts from simulation realism (Maigret, 1991). It is not plausible to temporarily freeze a dynamic battlefield while a static, global analysis is performed. A battlefield is a dynamic environment. For computer simulation to be realistic, the computer simulation of the battlefield should also be dynamic. Consequently, if a learning element which uses static environment data is added to the NPSNET AF, the learning element should operate in such a manner so as not to interrupt they system's dynamic combat simulation.

The need to tailor outputs in response to a changing environment requires that the NPSNET AF be a dynamic system. The actions taken by a dynamic system are based upon feedback from the environment. A dynamic system, therefore, requires environmental sensory input. It also needs an algorithm for determining future actions based upon the sensory data. Because a dynamic system relies upon sensory data, its concerns are generally localized to the area within reach of its sensors, and it generally limits its analysis efforts to localized, reactive planning (Maigret, 1991).

The NPSNET AF is a dynamic system that operates in an environment in which future decisions rely heavily upon environmental inputs. The survival of the autonomous agents within the NPSNET AF depends upon the agents' abilities to alter their performances based upon feedback from the environment. Because the system is dynamic, there is only limited time for analyzing past performance and altering it. Changes in performance must occur fairly rapidly. Consequently, local reaction to environmental stimuli is generally more important than is response planning. Planning in a dynamic environment is often a difficult problem and usually requires more time to perform than does a preplanned reactive response. Consequently, NPSNET AF agents execute preplanned responses in response to environmental stimuli instead of first trying to learn what the best response is. This means that the response they elicit may not be the correct response. Only a historical review of environmental reactions can verify the correctnesss of responses.

## B. REAL-TIME AND HISTORICAL LEARNING

Should machine learning take place as an after-action analysis or should it take place during the execution of a simulation? In real combat, learning

occurs during battle as well as after the battle is complete. Though we acknowledge that real-time learning is an integral part of battle, we believe that historical learning is a more fruitful approach for machine learning within the NPSNET AF. In a real-time learning environment, an autonomous agent analyzes the current environment to determine the best course of action. However, in a dynamic world, an action which seems appropriate at time *T* may not be appropriate at time *T+1* if the environment has changed signficantly. If this is the case, then the real-time learning mechanism needs to re-evaluate the environment and determine a new course of actions before it can react. This re-evaluation process can cause a system to delay responding to environemtal stimuli.

In a dynamic simulation such as within the NPSNET, prompt response in generally more important than is real-time learning. Since the NPSNET AF vehicle operate in reactive manner, looking only at local stimuli, delays in responding to environmental stimuli can have negative consequences. The autonomous vehicles within the NPSNET AF are involved in battle simulations. These agents need to be able to rapidly change their actions in order to survive (Kwak, 1992). We believe that reactive planning based upon historical learning will provide the agents with the ability to respond rapidly to their environment and still provide them with the ability to learn new behaviors over time.

# IV. ANALYSIS OF AUTONOMOUS FORCE MOVEMENT

In order for autonomous force realism to be accomplished, the AF must move as a real force moves. Our work focuses on improving the AF movement in three ways. First we provide the NPSNET AF vehicles the ability to move in line, column, left echelon, and right echelon formations. Second, we provide them with the capability of transitioning between formations. Last, we establish a rudimentary decision strategy for determining when to transition between formations. This strategy is dependent upon target location and terrain characteristics.

With respect to movement techniques and strategies, we look at both single and multiple agent movement and address items which affect transition strategies. The final transition strategy we propose is by no means elaborate or complete. Our intent is to build a basic transition strategy module upon which future improvements can be made. Our transition methodologies for changing between formations focus on simplicity. We use simple algorithms which provide a sense of realism.

Below, we discuss our concerns about AF movement and formation selection and transition strategies. After these discussions, we provide an in-depth explanation of the improvements we made to the NPSNET AF.

## A. SINGLE-AGENT MOVEMENT

Single-agent movement was the concern of autonomous force experts when the concept of computer generated forces was in its infant stages. With a single-agent, programmers did not worry about formation movements. Each agent acted independently of other agents, and steps were therefore required

to prevent collisions between agents. Generally, agents were provided with a mechanism which gave them the capability to sense their surroundings. However, single-agent movement was of limited use for military applications, since military forces tend to act in concert with numerous squadrons, platoons, divisions, companies, and brigades. Therefore, in order for computer generated forces to provide any significant value to the military, multiple autonomous agents moving in concert had to evolve.

## B. MULTIPLE AGENT MOVEMENT

When agents move together, three fundamental concerns arise. The first concern is how to keep the agents in a formation. The second concern is to determine when the unit should change formation. The third concern is to determine how to transition the agents from one formation to another. Culpepper (1992) provided us with the methodology for keeping the agents in a line formation. We provide the autonomous forces with various formation choices, the means to transition between formations, and a transition strategy.

## C. SELECTING THE BEST FORMATION

There are many factors which affect the choice of platoon formation. Some of these factors include the terrain and other environmental stimuli, the intended objective of the platoon, and the location, number, and type of enemy vehicles and weapons.

### 1. Movement of Agents in Response to Environmental Stimuli

In order for autonomous agents to fully realize their training value, they must, in one sense, act human-like. Consequently, they must respond to their environment. In order for agents to respond to their environment, they must be able to sense their environment and determine which perceptions are

important and which are trivial. To date, the autonomous forces in our work cannot sense their environment with respect to the terrain or the presence of objects such as roads and bridges. However, we identify certain factors which should be accounted for when building the capability to sense the environment into the autonomous forces project.

**2. Sufficient Sampling**

The method of sampling terrain data impacts upon the realism of a computer simulation. Terrain and environmental data for the NPSNET AF is stored in the NPSNET database. In order for an autonomous vehicle to be "aware" of its surroundings, it must read in data from a database. The vehicle may choose to read in all of the database or a certain portion of the database. The more data it reads, the greater the computer resources and time required by the agent. As the number of agents increases, these increased requirements may make reading of the entire database impractical and undesireable. Consequently, sampling of only portions of the database may be more prudent than reading in the entire database.

There are many different techniques for sampling a database, however, our concern is: "How much sampling is enough sampling?" If we are searching for a concrete item such as a tree, tank, or building, only limited sampling may be necessary. But, when we want to search for something more abstract, such as a hill, the question becomes more complex. What constitutes a hill will depend upon the resolution desired. If we can only see the world as a series of triangles that measure 100 meters on a side, then a hill, or at least part of a hill, can be determined by sampling the elevation at each of the three corners of the triangle to determine if one corner is higher

than any of the others. If one point is higher, then a hill, or at least part of a hill, exists. However, we still do not know definitively how large the hill is, nor do we know what the terrain looks like between the corners. How fine a resolution one uses will depend upon the computing resources and time available, since finer resolution requires greater resources and more processing time.

The sampling requirement, though, is not determined solely by the resolution needs of the autonomous forces. It is also affected by how far beyond the current agent location the autonomous forces need to sample. When one doubles the look-ahead distance from 8,000 meters to 16,000 meters, there is a corresponding quadrupling of the amount of terrain data which must be processed (from 64,000 square meters to 256,000 square meters). Also, how an autonomous force responds to the environment will depend upon how much of the environment the force can see. In the example above, the agents will most likely respond differently to the environment depending on whether they can see ahead 8000 meters or 16000 meters.

As we improve the AF, we are concerned with using computer resources economically. The more clearly we measure our surroundings, the greater is our need for computer resources. This need to clearly define our surroundings competes with the planned machine learning element of the NPSNET AF for computer resources. Sampling the environment too often, means having a crisp, clear, environmental picture at the expense of learning capabilities. Likewise, sampling too infrequently, allows more assets for learning, but it causes the picture of our world to be hazy and unrealistic.

### 3. Mission Objectives

The formation which autonomous agents travel in may be affected by their mission or task. Though current autonomous forces work identifies the mission at the start of the program, no provision exists to allow the user of the system, or the system itself, to change the mission. Since the actions of humans depend upon their objectives, and since one of the goals of autonomous forces is to more closely model the behavior and actions of humans, then it seems logical to suggest that the formation of multiple agents should depend upon the mission objectives of the agents.

### 4. Enemy Agents

Since one of the goals of our autonomous forces is to have them operate and survive in a hostile environment that contains enemy autonomous or semi-autonomous agents, we believe that the formation chosen by the autonomous forces should depend upon the number, location, and type of hostile enemy forces, in addition to the mission. Current capabilities of the autonomous forces program provide information on targets which are within a pre-defined attack range. The information does not distinguish between varying possibilities of enemies, and no rules exist within the rule base which make formation decisions based upon number and type of targets.

### 5. Selecting The Best Formation by Applying a Genetic Algorithm

We build into the NPSNET AF a basic, rule-based, formation selection strategy with fixed formation choices based upon specific inputs. The formation choices are line, column, left echelon, or right echelon. The inputs to the decision are the location of targets and a general terrain characteristic.

Currently there is no scoring mechanism within the NPSNET AF to measure the effectiveness of our formation choices. There is also no feedback on how a chosen formation affects the survivability of the AF. Implementation of a genetic algorithm to optimize formation decisions by assigning weights to various battlefield factors, relies on the use of a scoring mechanism. In Chapter V, we discuss a method for implementing a GA learning system within the NPSNET AF. This system can be used for determining which formation selection is best in a given situation. However, before it can do this, the AF system needs a technique for scoring the performance of the AF.

## D. MOVEMENT THROUGH A TURN

Culpepper's original AF (1992) maneuvers toward goals by moving a platoon *base point* and the platoon vehicles. Culpepper uses this arbitrary base point to control the movement of each of three platoons. Each tank within each platoon is offset from the base point. The base point serves as the reference point from which the agents within each platoon are offset. Culpepper places two agents to either side of the base point. The distance of these agents from the base point is fixed at 25 meters and 75 meters to the right and left of the base point. The base point for each platoon to the right and left of the center platoon is placed at a fixed distance from the base point of the center platoon, either (+200) meters or (-200) meters, depending on whether the platoon is to the right or left, respectively, of the center platoon. This results in all of the agents in each platoon being in a line formation as depicted in Figure 1.

When a turn is necessary, Culpepper's algorithm calculates the maximum possible turn and the maximum possible turning rate. A limiting system prevents the agents from exceeding either the maximum turn or the maximum turning rate. The algorithm assumes that the AF vehicles are in a line formation as depicted in Figure 1. The vehicles maneuver around the pivot point, offset a predetermined distance along the baseline of the line formation.

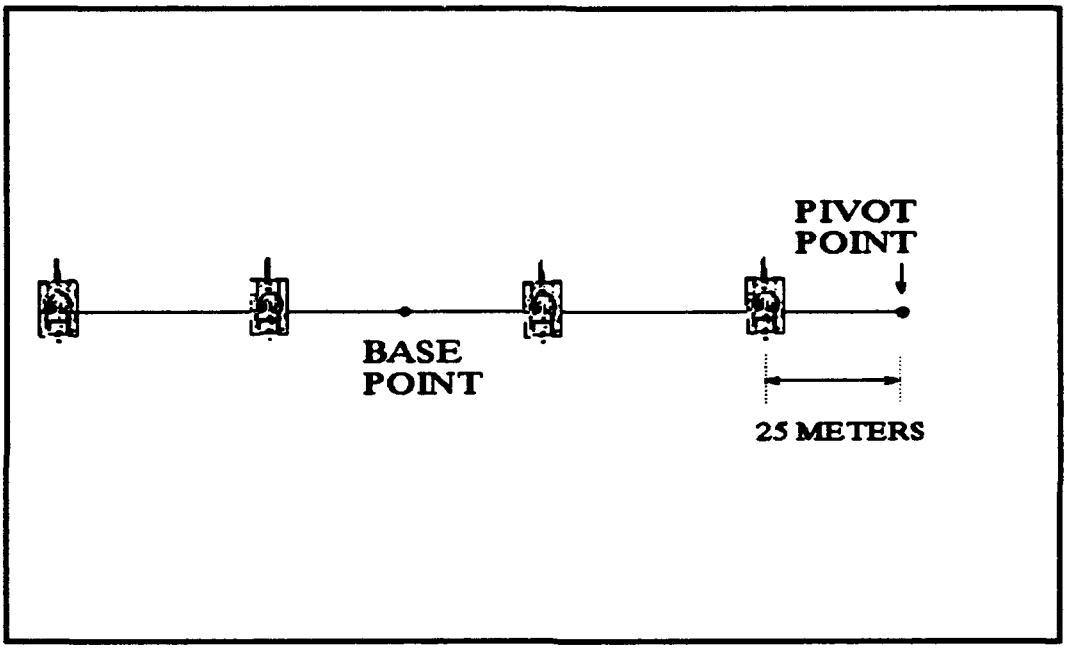



Figure 1. Line Formation With Base Point.

This algorithm works well for the line formation, but it is insufficient for other formations (left echelon, right echelon, and column). In these other formations, the turning distance of a vehicle and the speed it requires for the turn are affected by the echelon type and the direction of the turn. Therefore, we make the pivot point a dynamic point. Instead of placing it at a fixed distance from the outside vehicle, we allow its distance from the outside vehicle to vary, depending on the formation type. We also adjust the pivot

point so that it falls along the imaginary line running through the vehicles, regardless of the formation type.

## E. KEEPING AGENTS IN FORMATION

Two possible methods for maintaining agents in a formation are a) move each agent independently and keep track of its position relative to the position of the other agents, perhaps guiding off of one of the agents and b) move an arbitrary point through space and have the agents offset from this point. The first method allows for greater flexibility in movement and allows agents to more readily switch to independent operation. However, the greater independence also requires increased control over the agents, which translates into more involved and complex rule sets and computer code. The second method drastically constrains the freedom of each individual agent, but provides a means for easily maintaining the agents in formation. Culpepper (1992) uses the second method for maintaining his autonomous agents in formation.

Our efforts expand upon the work of Culpepper. We provide the capability for the agents to operate in and to transition between any one of the following formations:

- Line
- Right Echelon
- Left Echelon
- Column.

## F. FORMATION SELECTION

Deciding when to change formation, is a simple implementation of a decision matrix which can easily be expanded. As seen in Figures 2 and 3,

we made formation decisions as to the correct formation based on only two criteria: the nature of the terrain ahead of the agent, and the targets (if any) it faces.

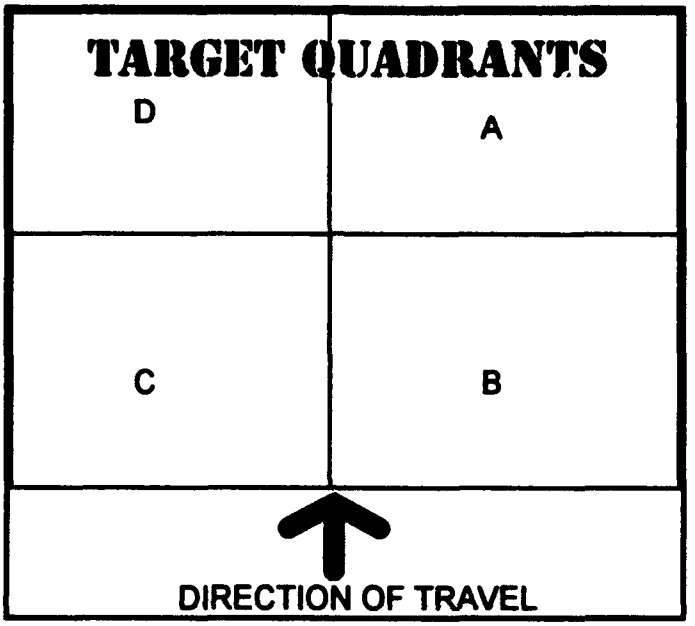



Figure 2. Target Quadrants and Direction of Travel.

| Decision Matrix For AF Formations | | |
|---|---|---|
| IF Terrain is as follows: | AND Targets are in the following quadrants: | THEN The recommended (new) formation is: |
| Narrow | None | Column |
| Narrow | A and D | If right echelon then stay there, else go to left echelon |
| Narrow | A only | Else go to left echelon<br>Right Echelon |
| Narrow | D only | Left Echelon |
| Wide | None | Current Formation (no change) |
| Wide | A and D | Line |
| Wide | A only | Right Echelon |
| Wide | D only | Left Echelon |

Figure 3. Decision Matrix for Formation Selection.

The direction of travel is described above as 000 or "North" between quadrants D and A. From Figure 3, one can see that if the terrain is narrow and there are no known targets to the front of the agent, then the recommended new formation is a column. If targets are present in quadrants D, A, or both, then the expert system directs a change to the left or right echelon, respectively. If the terrain is wide, then the overriding factor becomes the location of targets, with any formation possible in the absence of targets.

Since there is no current mechanism in the NPSNET AF for determining what constitutes narrow or wide terrain, and since manipulation of the terrain database to determine what constitutes a hill, ridge, finger, draw, etc., is beyond the scope of this thesis, we artificially generate the first factor (narrow or wide terrain) based on game cycles. However, we use current capabilities of the AF for target location information, making this a realistic and important input to the formation decision. When the mechanism is in place to glean high-level terrain information, as opposed to low-level terrain data values, from the database, then the terrain criteria may be easily expanded to include hills, rivers, roads, bridges, and impassable terrain, by simply adding rules (which we prototype) in the expert system rule base.

## G. THE ECHELON N FORMATION TRANSITION

In order for our multiple autonomous agents moving in formation to transition between allowed formations, we develop a formation transition algorithm, which we call the echelon n formation. The goal of the echelon n formation is to provide a relatively straight-forward method for transitioning between various autonomous platoon formations. Agents within our platoons

can be in any one of the following formations: a) line, b) column, c) left echelon, or d) right echelon.

When the agents within the platoon need to transition from one formation to another, two pieces of information are necessary. First, the system must know the current formation. Second, it must know the final desired formation. If we assume that these two pieces of information are available, then a transition algorithm from one formation to another is necessary.

One way to solve the transition dilemma would be to have a different transition algorithm for each transition. In other words, there would be a separate transition strategy for the transition between each formation X and each formation Y. Using this methodology, though, would require twelve different transition strategies, assuming we have the four possible formations of line, column, right echelon, and left echelon.

Another way to view the formations is to consider each formation as a special case of one particular formation. We call this one particular formation the echelon n formation. With the echelon n formation, we consider all formations to be a special case of an echelon formation. The "n" in the name "echelon n" is to annotate the angle above or below the horizontal. For example, a line formation is an echelon 0 formation. A left echelon formation is an echelon 45 formation. A right echelon is an echelon (-45) formation. Figure 4 illustrates the echelon n concept.

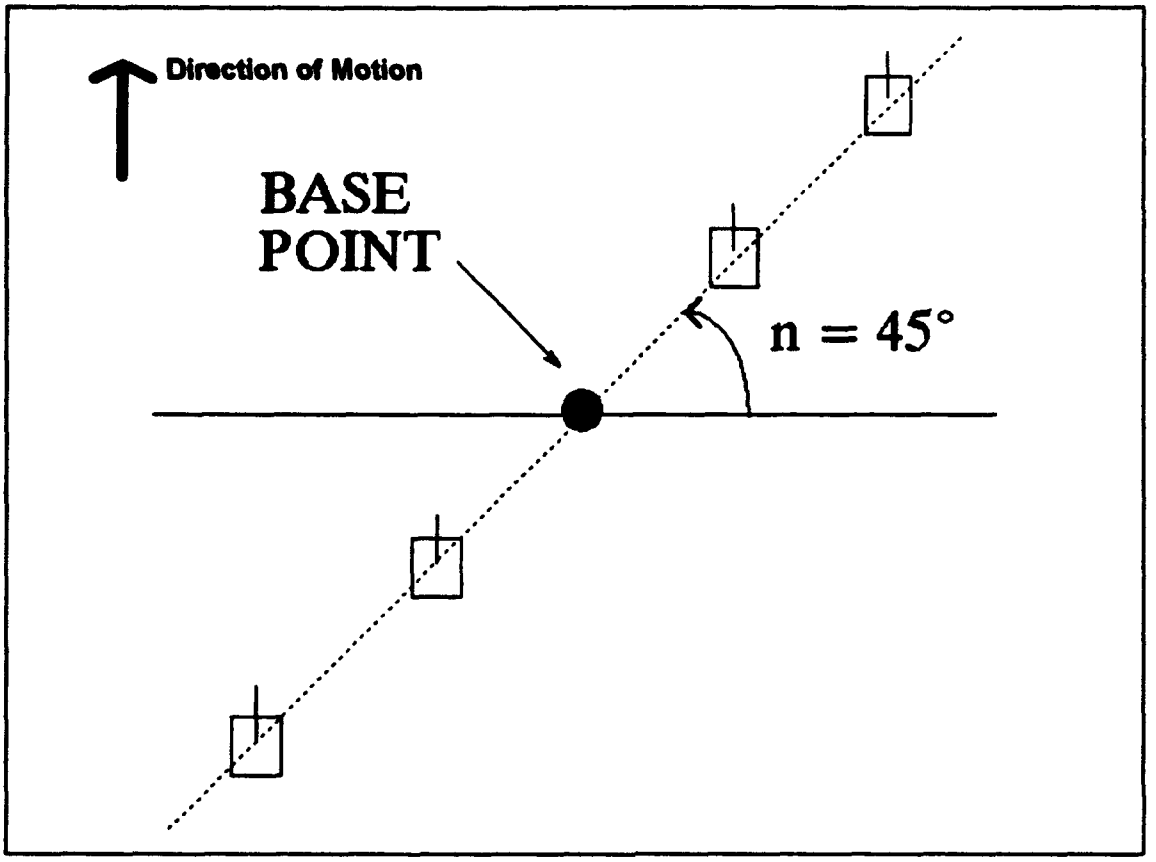



Figure 4. Echelon 45 Formation.

The motivation for using the echelon n formation is simplicity. By using the echelon n formation algorithm, instead of using twelve different transition formulas, we reduce the transition problem to a single algorithm. We choose simplicity over sophistication to keep the CPU-intensive algorithms to a minimum. In doing so, we reduce the likelihood of unnecessarily slowing down the simulation, since an unduly slow simulation provides little realism.

## H. REALISTIC TRANSITION

When agents move in formation, the goal of simulation is to make that movement look as realistic as possible. The same is true when agents transition from one formation to another. Therefore, when considering

formation changes, it is important to consider the rate of agent movement. Each agent must have sufficient speed to move down the path of intended movement and also be able to smoothly transition from one formation to another. The agents cannot be moving at their top rated speed prior to formation transition, since no agent would have sufficient remaining speed to make the transition. We address this concern by limiting the top speed of the agent in any formation to a value slightly less than that of the top rated unclassified speed of the agent. Given a top-rated speed of S meters per second and a limited speed of L meters per second, the excess speed, E, is:

$$E = S - L. \qquad \text{Equation 1.}$$

To calculate the excess speed available for formation transition in the autonomous forces program, we need to know the maximum speed of the M1A1 tank and the limiting speed imposed upon the tanks when moving in formation. Culpepper (1992) and Branley (1992) set the highest allowable speed of any tank moving in formation at 20 meters/second. We establish the top rated speed of the M1A1 tank to be 50 mph, which is equivalent to 22.4 meters/second. We calculate an excess speed of 2.4 meters/second which the tanks can use for formation transition.

Given an excess speed of E, we then estimate the maximum distance that each autonomous agent can transition during a given time frame. The time frame used by the autonomous forces is referred to as the average time. It is based upon the average time required to cycle through each run of the rule

base. We calculate the maximum possible transition distance, Tmax, as follows:

$$Tmax = E * \text{average time.} \qquad \text{Equation 2.}$$

Given Tmax, we determine how many iterations of the echelon n are necessary to ensure a smooth transition between formations. Thus, to go from echelon 0 to echelon 45, the autonomous vehicles step through the intermediate echelon values. How many intermediate steps are necessary depends upon the maximum distance that the outer most agents need to travel during a particular transition. This maximum distance of travel, Dmax, is calculated using the geometric relation between the angle of movement in radians, $\phi$, and the distance of an object from the center about which it is moving, which we call radius, R. The equation is:

$$Dmax \quad \phi \quad R \qquad \text{Equation 3.}$$

The minimum number of necessary iterations, Imin, is:

$$Imin = Dmax / Tmax. \qquad \text{Equation 4.}$$

Thus, if the average cycle time increases, then Tmax increases, which means that Imin decreases. If Tmax becomes large enough, then Imin becomes so small that the autonomous vehicles tend to jump from one formation to another instead of making a smooth transition. In order to assure a sense of realism, Tmax cannot grow without bound. Because the average cycle time is directly dependent upon the computational complexity of the underlying rule base, it is necessary to limit computationally intensive code in the rule base.

Therefore, our efforts reflect a trade-off between simplistically transitioning between two formations and transitioning in a manner which looks realistic.

## I. CONSTRAINING THE TURN

Because the AF can assume any of four possible formations, we update the turn-limiting algorithm developed by Culpepper (1992). He turned his autonomous vehicle platoons about a pivot point which was located a fixed distance from the outermost tank in the line formation. This distance was set at 25 meters and is illustrated in Figure 1. By using a pivot point, Culpepper could determine what the constraining turn would be and then limit the turn rate of the platoon to a value less than the limiting rate. His method, however, worked only for platoons operating in a line formation. We modify his pivot point and the limiting turn calculations so that we control the turn rate of the autonomous forces in any echelon n formation. Instead of placing the pivot point at a fixed distance from the platoon, we allow the pivot point to vary from a minimum of 25 meters up to a maximum of 50 meters. As depicted in Figure 5, we calculated the pivot point distance as a function of the echelon angle. Given an echelon angle of $\phi$, we establish a new pivot point using the formula:

$$\text{Pivot Point} = 25 \text{ meters} / \cos(\phi) \qquad \text{Equation 5.}$$

For an echelon 45, which is what we call our left echelon, the pivot point is 35.35 meters. As $\phi$ increases towards 90 degrees, the value for the pivot approaches infinity. Consequently, we limit the use of this particular constraining algorithm to the line, left echelon, and right echelon formations, as depicted in Figure 5.

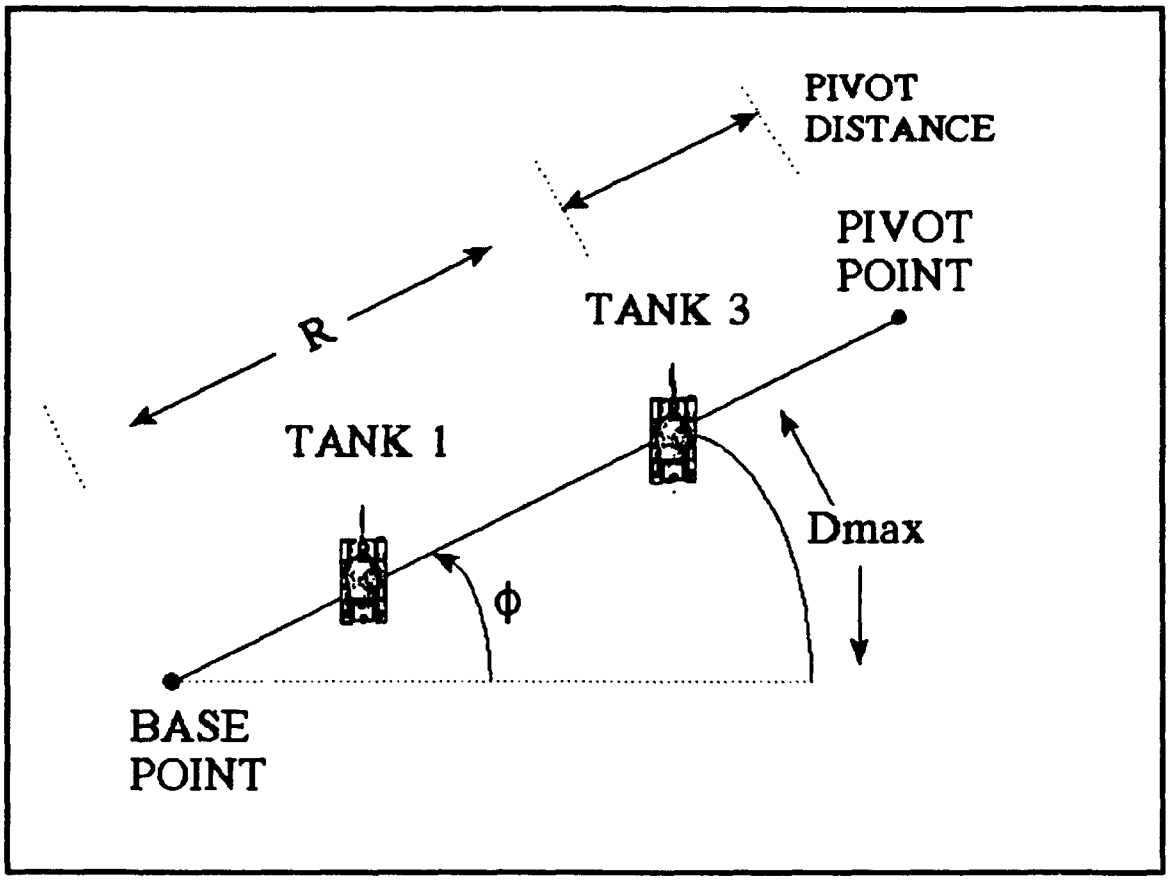



Figure 5. Pivot Point Distance.

# V. APPLICATION OF GENETIC ALGORITHMS TO AUTONOMOUS FORCES

## A. THE GENETIC ALGORITHM

### 1. Definition Of A Genetic Algorithm

What is a Genetic Algorithm? It is simply a method of programming a computer to simulate that which occurs in nature - survival of the fittest. That is, finding solutions to difficult problems through a computer-guided, probability-based search of the "landscape" of all possible answers to the problem at hand. In the words of John Holland (1992): "Computer programs that 'evolve' in ways that resemble natural selection can solve complex problems even their creators do not fully understand."

### 2. Reasons For Using A Genetic Algorithm

There are some problems whose optimal solutions are not solvable by most humans. In these problems a human can merely hope to gain a "good enough" solution. Often what prevents him from finding an optimal solution is an inability, due to a time constraint, to search all possible solutions. The implicitly parallel nature of a GA provides it with the ability to conduct a rapid, stochastic search through a vast number of possible solutions for the most likely optimal or "good enough" solution(Goldberg, 1989). Because the guided search technique that a GA uses can significantly reduce search times, a GA is a potentially powerful search methodology when most likely optimal or "good enough" solutions are sufficient.

## 3. The Mechanics Of A Genetic Algorithm

To conduct a search of the solution space for a given problem, a genetic algorithm selects possible solutions using selection, evaluation, and optimization methods similar to those used in nature. In nature, a random selection of parents produces offspring, and the offspring appear as a representation of their individual genetic codes or genotypes. This genetic makeup is stored on strips of DNA, called chromosomes. Each individual's phenotype is the physical result of a particular genotype and is simply "what the individual looks like," or perhaps "how the individual performs." Individual offspring may have arbitrarily undesirable traits such as being ugly, slow, or sickly; or, they may have more desirable traits such as being beautiful, powerful, healthy, and fast. In nature those individuals with desirable traits tend to survive, while those with undesirable traits tend to die off or at least grow fewer in number.

A GA copies from nature. A simple GA relies upon three operators known as selection, mutation, and crossover (Davis, 1991). As with nature, parents (binary strings of problem input values, for example) are selected and mated, and the characteristics of the parents are transferred to the children using crossover of the chromosomes which occurs stochastically. Sometimes, during mating, mutation occurs. Mutation of certain characteristics also plays a part in the evolutionary process by randomly changing a value along the chromosome and thereby allowing offspring to significantly differ from their parents. These differences may be either beneficial or detrimental. The beneficial effects tend to increase an individual's chance of survival, while the detrimental ones tend to reduce it. These three operators, selection, mutation,

and crossover, are used in the genetic algorithm much the way they occur in nature, and they are summarized in Figure 6.

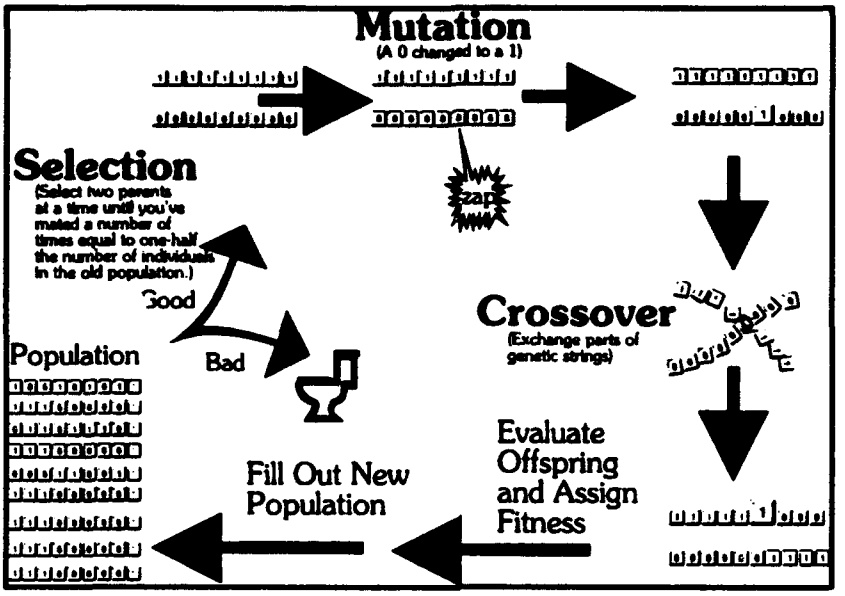



Figure 6. The Three Operators: Selection, Mutation, Crossover.

## 4. Details of the Functioning GA

We develop a fully functioning GA in the C programming language which is heavily patterned after Goldberg's Simple GA, written in Pascal (1989). We name ours INTEGER2, because it solves a backward search for two integers, which produce a certain known float value when one integer is divided by the other. We provide this GA for later use with the NPSNET AF. The source code for INTEGER2 is contained in Appendix A and can be obtained on disk through Professor Bhargava at the Naval Postgraduate School, code AS/BH. Sample performance characteristics of INTEGER2 are summarized in Appendix B.

The following discussion applies to the generic GA functions summarized in Figure 7.

1. Initialize the first population of individuals (chromosomes).
2. Evaluate the fitness of each individual. (EVALUATION)
3. Select the parents (in pairs) best suited to mate. (SELECTION)
4. Mate the parents (applying MUTATION & CROSSOVER in the process) and produce the new generation.
5. Repeat steps 2 through 4 until you run out of time.
6. When the time is up, return the best chromosome found (best solution to the problem).

Figure 7. Operational Steps Of A Simple Genetic Algorithm.

INTEGER2 follows a pattern similar to all genetic algorithms in its search. The program starts with a random sample of a population of individuals. For this discussion, we will assume that a population consists of 100 individuals, and each individual is represented by one chromosome of length eight. That is, there are eight positions along the chromosome which hold input values, and, collectively, the values on one chromosome constitute one solution to the problem under study. The position of each 1 or 0 along the chromosome is called a locus. Each chromosome is an individual, and there is no limit to the number of individuals in a population. However, for most implementations of the GA, this number is assumed to be evenly divisible by two (Goldberg, 1989).

This first step is initialization of the first population. At the very start of the program, the composition of each of eight loci along each of the chromosomes is determined using a function which generates random numbers. Although there is no such thing as an algorithm which generates

purely random numbers, the function we use to generate pseudo-random numbers in our GA provides us with sufficiently "random" numbers.

After the initial population is generated, the next step the GA follows is to evaluate all members of the current population to determine the fitness of each member. Fitness is a measure of how well a particular random solution satisfies an objective function. The fitness is the basis for order-ranking the individuals in the population (Goldberg, 1989).

Next, the GA makes a selection of the individuals to mate, producing offspring for the next generation, and these become the new population. As with nature, those individuals (chromosomes) with the highest fitness evaluations have the greatest chance of mating and producing the next generation, and those with the lowest fitness have little or no chance of producing members of the next generation. This selection process is one of three common "operators" (selection, crossover, and mutation) which act on each of the chromosomes in the population during reproduction (Goldberg, 1989).

After two parents are selected, they are mated. The next operator, mutation, comes into play in the mating process (Goldberg, 1989). The basic premise for this operator is that there is always a chance (usually a very small chance) that the mating process will have a random error, represented by changing a 1 to a 0, or vice versa. The basic reason for mutation is diversity. This operator can cause two good solutions to become pronouncedly bad in the next generation, and vice versa. The third operator in reproduction is crossover. When crossover occurs, it does so between the strings of two parent chromosomes during mating. The result is two child chromosomes

(Goldberg, 1989). Crossover, like mutation, is probabalistic, though it occurs much more frequently than mutation. If it does occur, two parent chromosomes exchange genetic material, and hence their genetic code, at a random exchange site, producing offspring which are made up of genetic material from each parent. Crossover may occur at one point along the two parent chromosomes, at multiple points, or not at all. Our GA uses single-point crossover with a probability of crossover equal to 50%.

**5. Tailoring the GA**

Genetic Algorithms can be enhanced to improve their efficiency. A more efficient GA tends toward the optimal solution more quickly than will a less efficient algorithm. However, there is a tradeoff for the increased efficiency. As the GA becomes tailored for the problem at hand, it tends to lose robustness (Davis, 1991). The solution set must be more clearly defined, and the ability to use the GA with different fitness functions is reduced. We use a technique known as selective reproduction (Koza, 1992) in our GA. Other GA tailoring methods include multi-point crossover, hybridization, and creep (Schultz and Grefenstette, 1992). We discuss selective reproduction as well as the other methods below.

Selective reproduction is a slight twist on the normal reproduction process. Instead of replacing all of the parent chromosomes with children using the crossover process, children are created directly from the parents. One way in which this is done is to select the two best parents from the population (those which generate the highest fitness function value) and then map each parent directly into a child. This ensures that the current "strongest" members survive. We use this mating scheme because it is

relatively simple to code and provides a relatively rapid optimization capability, while minimizing CPU requirements over other schemes we consider.

In the GA, survival is never guaranteed since the model is stochastic. Consequently, even though the strongest member of the population has the greatest likelihood of survival through reproduction, reproduction is not guaranteed. Reproduction is based upon a weighted probability, simulated through "roulette wheel" spins. Our version of selective reproduction alters this process by ensuring that the chromosomes of the two best parents are always passed onto the next generation.

Multi-point crossover is another process for enhancing the algorithm. As the name implies, instead of performing crossover at one point along the chromosome, more than one crossover point can be selected. For simplicity, we use only one crossover point for INTEGER2.

In a simple GA, the initial population from which the search for optima begins is usually selected at random. This is part of the inherent appeal of GA's. No knowledge of possible solutions is necessary. However, if some initial information is know, then it makes sense to use this information when selecting the initial population. For instance, if we know that positive numbers tend to work better than negative numbers, it is more prudent to select an initial population composed of only positive numbers. Placing negative numbers within the population only adds unwanted noise and increases the required search time to reach optima. This process of using known information about beneficial characteristics of possible solutions to help establish the initial population set is known as hybridization. Our

approach is a straight-forward randomization of the initial population and does not use hybridization.

Creep is similar to mutation. Whereas mutation can randomly flip any particular bit within a string, creep is limited to the lower order bits (Schultz and Grefenstette, 1992). For example, under mutation, the string 11000010 could be mutated to 01000010, which is a change from 194 to 66. If 194 is fairly close to optimal, then the optimum solution may be lost. With creep, however, 11000010 might only be changed to 11000011, which is a change from 194 to 195. Creep provides a means for fine tuning a solution. The INTEGER2 problem does not use creep, but we discuss it here because this concept may be useful for later implementations of the GA. Instead of creep, INTEGER2 relies on mutation, which is very important in preventing false solutions. A false solution could occur if the GA focuses its search around a local optima instead of a global optima.

## B. BENEFITS

An important benefit to developing autonomous forces capable of learning is to provide the user of the NPSNET simulator with an adversary which emulates the actions of a realistic opponent. With an intelligent AF, one human trainee can use the system without having to have another human driving the opposing force. But in order to make the simulation truly believable, the AF should be able to learn. Providing the AF with the ability to learn from its past actions is the primary benefit to be gained by using a genetic algorithm.

The GA is basically perceived as a tool for optimization or as a methodology for rapidly searching a solution space for a possible best

solution. It optimizes a search for a solution by remembering those characteristics of a set of solutions which makes it better than other solution sets. These characteristics are then passed on to the next generation. In other words, the program learns which solutions from the last generation produce the best outcomes. It then uses this information to guide its search in the next generation as it searches for possible solutions.

A battlefield commander certainly learns as the battle progresses. Some of the commander's actions are based on reaction, and this is analogous to the pre-programmed script some autonomous forces play out. However, the commander may try an action which defies established procedures and previous experience. With the AF, this is where the expert system, alone, falls short, and the GA may be useful. Implementing the GA allows the AF to try new approaches and learn from them, as opposed to carrying out only stimulus-response actions to pre-programmed decision rules.

By using a machine learning methodology, it is no longer necessary to pre-program all possible behaviors (stimulus-response rules). Instead, a system capable of machine learning can learn new reactive behaviors. This reduces the amount of pre-programmed information required from domain experts and reduces the knowledge acquisition bottleneck (Schultz and Grefenstette, 1992).

## C. LIMITATIONS

We discover that a GA has the potential for inordinately slowing down any real-time computer simulation if placed in series with an expert system which interacts with a dynamic environment. Applying a GA to the NPSNET AF can possibly degrade the performance and believability of the simulation.

Specifically, if a GA consumes excessive computer resources as it searches for a near-optimal problem solution, it can interrupt the normal flow of vehicle information and thereby degrade the realism of the computer simulation.

## D. APPROACHES

### 1. Our Approach

We develop a stand-alone simple genetic algorithm which uses selective reproduction to optimize an answer to a generic algebraic problem. Although the objective or fitness function of INTEGER2 is a simplistic one, it demonstrates the ability of the algorithm to learn better solutions from generation to generation of program execution. INTEGER2 can be easily integrated into the NPSNET AF once a mechanism to support historical learning through fitness evaluation is added to the AF system.

### 2. Reason For Delaying Integration Of The GA With NPSNET

The current state of development of the NPSNET AF is depicted in Figure 8. This figure shows the relationship between the NPSNET AF expert system and the execution code.

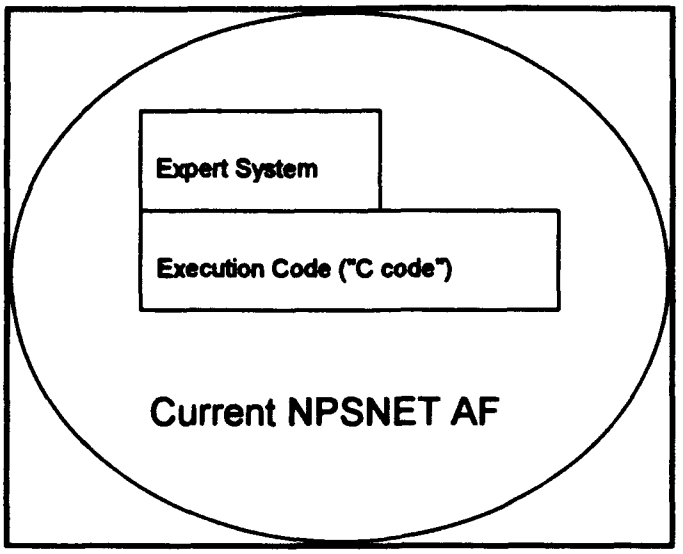



Figure 8. Current NPSNET AF Hierarchy.

We attempted to implement our GA in series with the expert system and the execution code. Our attempted approach is depicted in Figure 9.

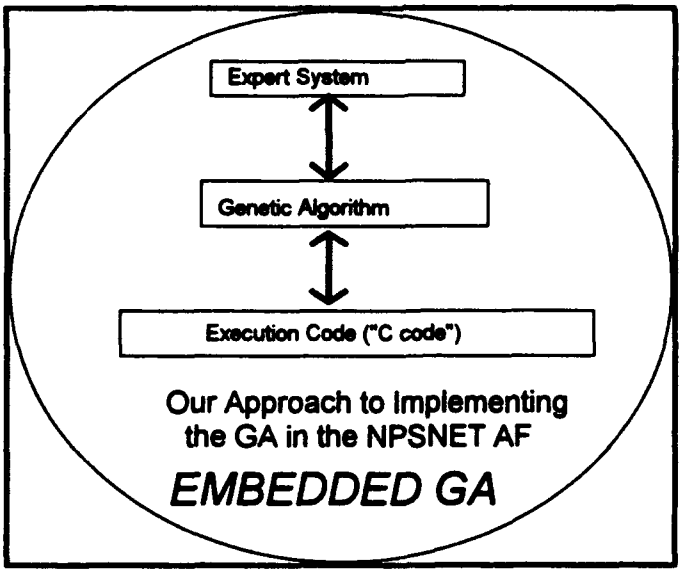



Figure 9. NPSNET AF Embedded GA Hierarchy.

We believe that a better way for the GA to work with the NPSNET AF without disrupting the dynamic performance of the AF agents is to allow the GA to work independently from, and in parallel with, the expert system, as shown in Figure 10 below. Using this method, the GA and expert system can communicate via messages. In doing so, the GA acts as a truly independent player program which sends messages to the AF expert system. A GA can take considerable time to reach a solution when searching a complex solution space. By making the GA an independent, parallel learning system, however, regardless of how much time the GA takes to conduct machine learning, the expert system continues to operate, using its most recent information from the GA.

One potentially effective GA implementation hierarchy is displayed in Figure 10 below. In Figure 10, the execution system (NPSNET AF - left side of diagram) cannot be slowed by occasionally long processing cycles of the learning system (GA-right side of diagram) (Grefenstette and Ramsey, 1992).

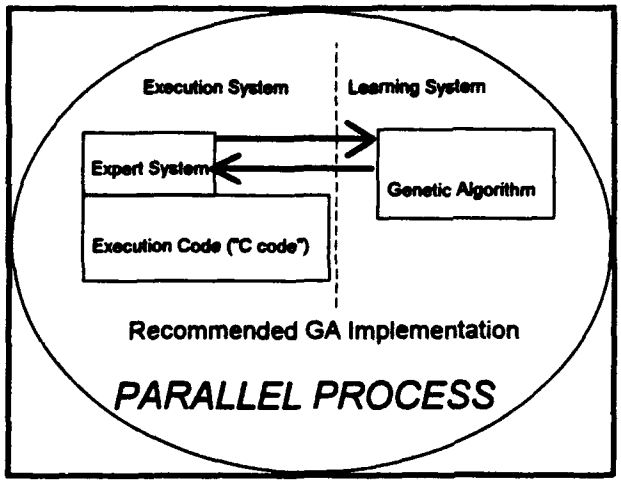



Figure 10. Recommended NPSNET AF Hierarchy.

In Figure 10, the expert system and operational code of the NPSNET AF, together, comprise the execution system. The GA is the learning system. We feel that the GA on the execution side of the NPSNET AF could slow down the simulation and diminish realism. Grefenstette and Ramsey (1992) find that a dynamic execution system which operates in a changing environment cannot be routinely disrupted to allow static or historical learning to take place. A dynamic system is, by its very nature, reactive. It responds to its environment. Its execution speed must keep pace with the tempo of environmental changes. If a dynamic system does not keep abreast of environmental changes, it cannot compute appropriate reactions in response to the current environmental stimuli (Paquet and Lamontagne, 1993). Agents such as those within the NPSNET AF will most likely fail in their efforts if they cannot accurately respond to environmental situations.

# VI. CONCLUSION

Useful combat simulation requires simulation that is realistic. When autonomous forces are used within the simulation, their actions and behavior should be human-like. They should be capable of learning new techniques from their experiences. Our goal is to improve the NPSNET AF so that its autonomous vehicles operate more realistically.

We look at improving the sense of realism within the NPSNET AF in two areas. First we attempt to provide the AF with a machine learning capability by trying to integrate a GA into the NPSNET AF. Second, we enrich the underlying rule base of the NPSNET AF. Our changes include providing the AF with a choice of four possible formations, providing it with the echelon n mechanism for transitioning between formations, and by providing the NPSNET AF with a decision strategy for deciding which formation in which to place the AF.

Though we develop a functional GA, called INTEGER2, we do not integrate it with the NPSNET AF for the following reason. Embedding the GA into the current expert system in a serial manner is disruptive to the normal control flow between the expert system and the execution code of the NPSNET AF. Originally, we built INTEGER2 to provide dynamic, real-time learning. Our intent was to place INTEGER2 in series between the expert system and the execution code of the AF, but this approach did not support the dynamic environment of the NPSNET AF. A potentially better approach is to implement the GA as an independent, parallel process that is capable of

conducting historical learning and routinely updating the NPSNET AF expert system, without degrading the performance of the execution system.

As the NPSNET AF matures, we see four possible areas for further work. These areas are:

- Implement a GA as an independent, parallel learning system
- Enrich data input/output
- Build an AF scoring algorithm
- Document the NPSNET AF to assist future programming efforts

In order for the NPSNET AF system to learn, it needs a learning mechanism. A genetic algorithm is one such mechanism. If placed in an independent, parallel hierarchy, it can potentially provide historical learning which will improve the NPSNET AF expert system's ability to react to its environment. Another problem for which INTEGER2 might prove adept is that of artillery registering of fires. In this problem, multiple guns have multiple munition types with multiple stockpiles to lay on multiple targets. This is a good historical learning problem because it is a linear programming dilemma in which the results are easy to measure and the problem is sufficiently difficult to warrant computer assisted optimization.

The current NPSNET AF system knows relatively little about the environment in which it operates. A major reason for this is that the AF receives almost no environmental information from the NPSNET. The information it receives concerns the status of other vehicles, but no information is provided about terrain or obstacle characteristics. Before the AF can make better decisions, it must be capable of accepting data about the world in which it operates.

One concern we have with the NPSNET AF is that the current system provides no means for measuring AF performance and assigning credit for that performance to a rule or particular set of rules. In order for a GA to pick the best solution or rule strategy, the ability must exist to assign credit to the rules which lead to better strategies. Before credit can be assigned, a performance measure and the ability to correlate the performance to a rule or set of rules are necessary.

Lastly, as the NPSNET AF grows in complexity, documentation of its capabilities and structure will minimize development problems and provide continuity between the efforts of individuals working on the system (Page-Jones, 1988).

## APPENDIX A

```
/*  FILENAME: A:\SGA\INTEGER2.C        */
/*  John Steiner and Robert Jacobs     */
/*  Curriculum code: 37                */
/*  Section: PM21                      */
/*  23 August, 1993                    */
/* Description: This is a genetic algorithm which conducts a probabalistic
search  for two unknown integers which, when the first is divided by the
 second, produced a known  float Target value such as 3.61538, for example
(user enters this).  */
/* This GA is patterned after Dr. D. E. Goldberg's SIMPLE GA, coded in
PASCAL,  as described in: Goldberg, David E., Genetic Algorithms in
Search, Optimization, and Machine Learning, Addison-Wesley Publishing
Co., Reading, Mass, 1989.  */
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
/* #define RAND_MAX 32767 */
#define MAXPOP 50    /* THE DESIRED POPULATION MUST BE
EVEN-NUMBERED*/
#define COLS_DESIRED 8 /* SPECIFIES NO. OF CHROMOSOME
LOCI*/
#define SEED 45
#define INITIAL 0
#define PCROSS 0.0    /* NORMALLY .5 EXCEPT FOR THIS
APPLICATION */
#define MAXGEN 2000

/* FUNCTION DECLARATIONS ZZZZZZZZZZZZZZZZZZZZZZZZZZZ*/
float statistics();
float initialize();
float decode();
float chdecode();
float objfunc();
int generation();
```

```
int select ();
void genstats();
void popstats();

/* TYPE DECLARATIONS TTTTTTTTTTTTTTTTTTTTTTTTTTT */
struct chromosome_type
    {int CHROM[COLS_DESIRED+4];}
    CHROMOS,MATE1,MATE2,CHILD1,CHILD2, *p, *p1, *m1, *m2,
    *c1, *c2;

typedef struct { int CHROMO[COLS_DESIRED+4];
        float X;
        float FITNESS;
        int PARENT1;
        int PARENT2;
        int XSITE;
        }individual_type;

typedef struct {individual_type POP[MAXPOP+4];}population_type;


/* INSTANTIATIONS OF TYPES IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII*/
individual_type  INDIV, INDIV2, INDIV3, INDIV4, *q, *q2, *q3, *q4,
*qtest;
population_type OLDPOP, NEWPOP, TEMPPOP, *op, *np, *tp;

/* VARIABLES FOR MAIN AND GLOBAL USAGE VVVVVVVVV */

int COL_NUM = 0;  /* COLUMN COUNTER VARIABLE FOR ARRAYS
*/

int ROW_NUM = 0;  /* ROW COUNTER VARIABLE FOR ARRAYS */

int POPSIZE=10; /* TO KEEP TRACK OF THE CURRENT SIZE OF THE
POPULATION */
```

```
int LCHROM=7;  /* ANOTHER VARIABLE FOR CHROM. LENGTH --
SEE ALSO COLS_DESIRED*/

int GEN=0;          /* TO KEEP TRACK OF THE CURRENT
GENERATION */

int NMUTATION=0; /* INITIAL VALUE FOR THE NUMBER OF
MUTATIONS / GENERATION */

int NCROSS=0;   /* INITIAL VALUE FOR THE NUMBER OF CROSSES
PER GENERATION */

int JCROSS=0;   /* COUNTER FOR THE CROSSOVER POSITION */

int SELPAR1;       /* A COUNTER VARIABLE FOR THE PARENT
SELECTION ROUTINE */

int SELPAR2;       /* A COUNTER VARIABLE FOR THE PARENT
SELECTION ROUTINE */

int POPMUTATIONS=0; /* THE NUMBER OF MUTATIONS IN TOTAL
FOR THE POPULATION */

int POPCROSSES=0;  /* THE NUMBER OF CROSSOVERS FOR THE
WHOLE POPULATION */

int POPMAXGEN=0;

int POPMINGEN=0;

int k=0;  /* COUNTER VARIABLE FOR GEN. STATISTICS
(GENSTATS)  */

int CHILD_NO;

int SHOWME = 1; /* set to 0 if you want to see generational stats only
```

```
        anything other than 0 will give you verbose performance

        infomation */

int MAX_PARENT = 0;    /* counter to keep track of the highest-fit parent*/

int NEXT_MAX_PARENT=0; /* counter to keep track of the 2nd highest
parent*/

float TARGET = 0.0;  /* THE TARGET FLOAT VALUE FOR WHOSE
PARENTS WE SEARCH */

float CEILING = 512.000; /* should be col_num raised one more power of 2
*/
float MAXMIN [MAXGEN] [6]; /* Array to hold MAX, MIN, AVG,
LIKELY_PAR1, LIKELY_PAR2 for each gen */

float PMUTATION = 0.00;  /* THE GLOBAL PROBABILITY OF
MUTATION,

NORM=0.03 */

float SUMFITNESS = 0.0; /* INITIAL VALUE FOR SUM OF EACH
GEN.'S FITNESS */

float X = 0.0;         /* VALUE OF EACH CHROMOSOME PRIOR TO
OBJ.

FUNCTION */

float FITNESS = 0.0;    /* THE FITNESS VALUE OF EACH
CHROMOSOME AFTER OBJECTIVE FUNTION */

float AVG=0.0;         /* AVERAGE FITNESS FOR ALL CHROMS. IN A
GENERATION */
```

```
float MAX=0.0;          /* MAXIMUM FITNESS FOR ALL CHROMS.
IN A GENERATION */

float NEXT_MAX=0.0;

float LIKELY_PAR1 = 0.2; /* The X value of the most likely parent #1
            don't set this to 0.0 */

float LIKELY_PAR2 = 0.2; /* The X value of the most likely parent #2
            don't set this to 0.0*/

float BEST_POP_PAR1 = 0.0; /* Stores highest X value of LIKELY_PAR1
from  generation to generation */

float BEST_POP_PAR2 = 0.0; /* Stores highest X value of LIKELY_PAR2
from generation to generation */

float MIN=0.0;      /* MINIMUM FITNESS FOR ALL CHROMS. IN A
GENERATION */
float POPMAX=0.0;
float POPMIN=0.0;
float MATING_DOMINANCE = 0.2; /* frequency at which fittest parents
will be forced to mate for the next generation */

time_t t;  /*  Initialize variable "t", which will be taken
      from the O/S time clock */

/*  MMMMMMMMMMM MAIN FUNCTION MMMMMMMMMMM */

void main ()
{

srand((unsigned) time(&t));  /* initialize the random number generator */
printf("\nPlease enter the probability of mutation (float value only):  ");
scanf("%f", &PMUTATION);
```

```
printf("\nYour selected value was:  %4.2f", PMUTATION);
printf("\nEnter '1' to view chromosomes and stats. 'O' for summary only.");
scanf("%d", &SHOWME);
printf("\nEnter the target float value:   ");
scanf("%f", &TARGET);

op=&OLDPOP;
np=&NEWPOP;
GEN = 0;

/* RR Remove Intialization of DOS specific random function RR */
/*  randomize ();  RRRRRRR  */

initialize(GEN);

while (GEN < MAXGEN)
    {
    GEN++;
    generation();
    if(SHOWME) printf("\n\n");
    }
popstats();

}

/* IIIIIIIIIIIIIIIIIIIIIIII  INITIALIZE IIIIIIIIIIIIIIIIIIIIIIIIIII */

float initialize(GEN)
int GEN;

{

int RAND_NUM; /* random number between 0 and 100 */
int ALLELE;  /* value at each locus on the chromosome: either 1 or 0 */
```

```
p=&CHROMOS; /* ASSIGN THE chromosome_type POINTER TO 1ST
POS. OF CHROMOS */

q=&INDIV;  /* ASSIGN THE individual_type POINTER TO 1ST POS. OF
INDIV */

op=&OLDPOP;
np=&NEWPOP;
qtest=&INDIV;

header(GEN); /* Set up the initial printout information */

/* GENERATE MATRIX AND FILL OUT POPULATION */
for (ROW_NUM=0; ROW_NUM < MAXPOP; ROW_NUM++)
    {
    if(SHOWME) printf ("%d\t", (ROW_NUM+1));
    for (COL_NUM=0; COL_NUM < COLS_DESIRED;COL_NUM++)
        {
        /* GENERATE RANDOM NUMBER 0<X<100  WITH FAIR COIN
TOSS*/
/* RRRRRR Remove DOS specific random and replace with ANSI specific
rand() RRRRRRRR */

/* RRRRRR if (random(100) < 50) ALLELE = 0; RRRRRR */

        if ((rand() % 100) < 50) ALLELE = 0;
        else ALLELE = 1;
/* FILL OUT CHROMOSOME WITH RANDOM 1'S AND 0'S */

        p->CHROM[COL_NUM] = ALLELE;
        q->CHROMO[COL_NUM] = p->CHROM[COL_NUM];
        if(SHOWME) printf ("%d ", q->CHROMO[COL_NUM]);
        }
    q->X = decode(0, COLS_I  .iRED);
    q->FITNESS = 1.0; /* First generation parents will have no
                         real fitness value, but this cannot be
```

```
                0.0 to prevent divide by zero error */
      /* q->FITNESS = objfunc(q->X);  */
    q->PARENT1=0;
    q->PARENT2=0;
    q->XSITE=0;
    op->POP[ROW_NUM]= INDIV; /* population[j] gets INDIV */
       if(SHOWME) printf("\t%9.5f ",q->X);
       if(SHOWME) printf("%10.5f ", q->FITNESS);
       if(SHOWME) printf("\t%d   ", q->PARENT1);
       if(SHOWME) printf("\t%d   ", q->PARENT2);
       if(SHOWME) printf("\t%d \n", q->XSITE);
    }
 statistics(MAX, MIN);  /* PASS MAX AND MIN AND RECEIVE
SUMFITNESS */
 return 69.0;
 }

/*  HHHHHHHHH  HEADER FUNCTION HHHHHHHHHH */

int header (GEN)
int GEN;
{
/* PRINT HEADER AND INITIAL INFORMATION */
if(SHOWME) printf("\t\tThis is generation:  %d\n", GEN);
if(SHOWME) printf("Number of Individuals per generation = %d\n",
MAXPOP);

if(SHOWME) printf("Number of Alleles per chromosome = %d\n",
COLS_DESIRED);

if(SHOWME) printf("Number of Generations Requested = %d\n",
MAXGEN);

if(SHOWME) printf("Probability of Mutation per Allele = %1.4f\n",
PMUTATION);
if(SHOWME) printf("CHROM#\t");
```

```
if(SHOWME) printf("CHROMOSOME\t\t");
if(SHOWME) printf("X VALUE
FITNESS\t\tPARENT1\tPARENT2\tXSITE\n");

return 69;

}

/* DDDDDDDDD DECODE FUNCTION DDDDDDDDDDD */
float decode(i,j)
int i;
int j;

{
float X = 0.0;
float POWER_OF_2 = 1.0;
p = &CHROMOS;
 /* p1= &CHROMOS + COLS_DESIRED;  */
q = &INDIV;

while (i < j)
    {
    if(p->CHROM[j-1]==1)  /* if allele is a 1, increment pwr of 2 */
        {
        X = X + POWER_OF_2;
        }
    POWER_OF_2 = POWER_OF_2 * 2;
    j--;
    }
return X;
}

/* OOOOOOOOO OBJECTIVE FUNCTION OOOOOOOOO */
float objfunc(SELPAR1, SELPAR2)
int SELPAR1, SELPAR2;
{
```

```
float SELPAR1_X, SELPAR2_X;
/* test */
float PARENT_RATIO = 0.0;

op = &OLDPOP;
INDIV3 = op->POP[SELPAR1];
INDIV4 = op->POP[SELPAR2];
q3 = &INDIV3;
q4 = &INDIV4;

SELPAR1_X = q3->X;
SELPAR2_X = q4->X;

/* test
printf("\nSELPAR1_X is:  %9.5f\n ", SELPAR1_X);
printf("\nSELPAR2_X is:  %9.5f\n\n ", SELPAR2_X); */

/* error trap */
if (SELPAR2_X == 0.0)
    {
    PARENT_RATIO = 77.00;
    printf("SELPAR2_X WAS 0.0"); /* TEST */
    }
else
    PARENT_RATIO = (SELPAR1_X / SELPAR2_X);
/* test
printf("\nPARENT_RATIO (1_X / 2_X) is:  %9.5f ", PARENT_RATIO);
printf("\nTARGET is:  %9.5f\n\n ", TARGET); */

if (PARENT_RATIO >= TARGET)
    FITNESS = (CEILING - (PARENT_RATIO - TARGET));
  else
    FITNESS = (CEILING - (TARGET - PARENT_RATIO));
/* TEST
printf("\nFITNESS is:  %9.5f\n\n ", FITNESS); */
/* FITNESS = X*X; */
```

```
/* FITNESS = (X*X*X*X*X)-(10.0*X*X*X*X)+(20.0*X*X*X)-
(4.0*X*X)+10000.0; */
 return FITNESS;

}
/* SSSSSSSSSSSSSS STATISTICS SSSSSSSSSSSSSSSSSSSS */
float statistics(MAX, MIN)
float MAX;
float MIN;
{
int j=0;
int MOM = 7;
int DAD = 7;
op = &OLDPOP;
tp = &TEMPPOP;
/* INITIALIZE THE VARIABLES WITH THE FIRST ELEMENT OF THE
ARRAY */

SUMFITNESS     =     op->POP[j].FITNESS;
MIN            =     op->POP[j].FITNESS;
MAX            =     op->POP[j].FITNESS;
MAX_PARENT = 0;
NEXT_MAX = -5.5;  /* -5.5 used to show errors - could also be 0.0 */
NEXT_MAX_PARENT = 0;

/* START SUMMING FROM THE SECOND ELEMENT OF THE
ARRAY */
for (j=1; j < MAXPOP; j++)
    {
    SUMFITNESS = SUMFITNESS + op->POP[j].FITNESS;
    if(op->POP[j].FITNESS >= MAX)
        {
        NEXT_MAX = MAX;
        NEXT_MAX_PARENT = MAX_PARENT;
        MAX = op->POP[j].FITNESS;
        MAX_PARENT = j+1;
```

```
        MOM = op->POP[j].PARENT1;

        /* TEST
        printf("\n\nMom is:  %d ", MOM); */

        DAD = op->POP[j].PARENT2;

        /* TEST
        printf("\n\nDAD is:  %d ", DAD); */

        LIKELY_PAR1 = tp->POP[MOM].X; /*FIRST LIKELY X VALUE
*/
        LIKELY_PAR2 = tp->POP[DAD].X; /*SECOND LIKELY X
VALUE */
        }
    if(op->POP[j].FITNESS > NEXT_MAX && op->POP[j].FITNESS <
MAX)
        {
        NEXT_MAX = op->POP[j].FITNESS;
        NEXT_MAX_PARENT = j+1;
            }
    if(op->POP[j].FITNESS < MIN) MIN = op->POP[j].FITNESS;
    }
AVG = SUMFITNESS/((float)MAXPOP);
genstats(GEN, MAX, MIN, AVG, LIKELY_PAR1, LIKELY_PAR2);
if(SHOWME) printf("\nThe AVG for the generation was:  %9.5f", AVG);

/*TEST*/
if(SHOWME) printf("\nThe MAX was:      #%d, %9.5f", MAX_PARENT,
MAX);

/*TEST*/
if(SHOWME) printf("\nThe NEXT_MAX was: #%d, %9.5f",
NEXT_MAX_PARENT,
NEXT_MAX); /*TEST*/
```

```
if(SHOWME) printf("\nThe MIN at the end of statistics loop was:  %9.5f",
MIN);
/*TEST*/
if(SHOWME) printf("\nThe Most Likely Parent 1 was:  %9.5f",
LIKELY_PAR1);
/*TEST*/
if(SHOWME) printf("\nThe Most Likely Parent 2 was:  %9.5f",
LIKELY_PAR2);
/*TEST*/
if(SHOWME) printf("\n\n\n\n"); /*TEST*/
return SUMFITNESS;
}

/*  GGGGGGGGGGGGGGG  GENERATION  GGGGGGGGGG */
/* THIS FUNCTION ASSUMES AN EVEN-NUMBERED POPULATION
*/
int generation()

{
int j = ROW_NUM = 0;
int COL_NUM=0;

c1 = &CHILD1;
c2 = &CHILD2;
q  = &INDIV;
q2 = &INDIV2;
op = &OLDPOP;
np = &NEWPOP;
tp = &TEMPPOP;

while (j < MAXPOP)
    {
    if(j < (int)MAXPOP * MATING_DOMINANCE)
        {
        SELPAR1= MAX_PARENT-1;
        SELPAR2 = NEXT_MAX_PARENT-1;
```

```
        }
    else
        {
        SELPAR1 = select();
        do
                SELPAR2 = select();
        while ((SELPAR2 == SELPAR1) || (op->POP[SELPAR2].X == 0.0));
/* KEEP TRYING TO SELECT PARENT 2 UNTIL YOUR ANSWER IS
OTHER
THAN PARENT ONE AND AS LONG AS THE PARENT 2 HAS AN X
VALUE
NOT EQUAL TO ZERO */
        }
    JCROSS=crossover(SELPAR1, SELPAR2);
        COL_NUM=0;
        if(SHOWME) printf("\nCHILD%d IS:  ", j+1);  /* TEST */
        for(COL_NUM=0; COL_NUM < COLS_DESIRED; COL_NUM++)
                {
                q->CHROMO[COL_NUM] = c1-CHROM[COL_NUM];
                if(SHOWME) printf("%d", q->CHROMO[COL_NUM]);
                }
        q->X = chdecode(1);  /* CALL TO CHDECODE FOR CHILD ONE
*/
        q2->FITNESS =  objfunc(SELPAR1, SELPAR2);
        /*  q->FITNESS = objfunc(q->X); */
        q->PARENT1=  SELPAR1;
        q->PARENT2=  SELPAR2;
        q->XSITE=JCROSS;
        np->POP[j] = INDIV;
        tp->POP[j] = op->POP[j];

                if(SHOWME) printf("\t%9.5f ",q->X);
                if(SHOWME) printf("%10.5f ", q->FITNESS);
                if(SHOWME) printf("\t%d   ", (q->PARENT1)+1);
                if(SHOWME) printf("\t%d   ", (q->PARENT2)+1);
                if(SHOWME) printf("\t%d   ", q->XSITE);
```

```
        if(SHOWME) printf("\nCHILD%d IS:  ", j+2);  /* TEST */
        COL_NUM=0;
        for(COL_NUM=0; COL_NUM < COLS_DESIRED; COL_NUM++)
                {
                q2->CHROMO[COL_NUM] = c2->CHROM[COL_NUM];
                if(SHOWME) printf("%d", q2->CHROMO[COL_NUM]);
                }
        q2->X = chdecode(2);  /* CALL TO CHDECODE FOR CHILD TWO
*/
        q2->FITNESS =  objfunc(SELPAR1, SELPAR2);
        /* q2->FITNESS =  objfunc(q2->X); */
        q2->PARENT1=SELPAR1;
        q2->PARENT2=SELPAR2;
        q2->XSITE=JCROSS;
        np->POP[j+1] = INDIV2;
        tp->POP[j+1] = op->POP[j+1];
                if(SHOWME) printf("\t%9.5f ",q2->X);
                if(SHOWME) printf("%10.5f ", q2->FITNESS);
                if(SHOWME) printf("\t%d   ", (q2->PARENT1)+1);
                if(SHOWME) printf("\t%d   ", (q2->PARENT2)+1);
                if(SHOWME) printf("\t%d \n", q2->XSITE);
    /* RESET THE PARENT VARIABLES FOR THE NEXT SELECTION
*/
    SELPAR1 = 0;
    SELPAR2 = 0;
    j = j+2;
    }

/* ADVANCE THE GENERATION */
for (ROW_NUM=0; ROW_NUM < MAXPOP; ROW_NUM++)
    {
    op->POP[ROW_NUM] = np->POP[ROW_NUM];
    }
statistics(MAX, MIN);

return 69;
```

```
}

/* SSSSSSSSSS SELECT FUNCTION  SSSSSSSSSSSSSS */

int select ()
{
float RAND=0.0;
float PARTSUM=0.0;
int SELVAL=0;
op= &OLDPOP;
/* RRRRRR Replace the DOS specific random with ANSI rand(). */

/* RRRRRR RAND = ((((float)random(100))/100.00) * SUMFITNESS);
RRRRRRR */
RAND = ((((float)rand())/((float)RAND_MAX)) * SUMFITNESS);

for(SELVAL=0; (SELVAL+1 < MAXPOP) && (PARTSUM < RAND);
SELVAL++)
    {
    PARTSUM = PARTSUM + op->POP[SELVAL].FITNESS;
    }
PARTSUM = 0.0;
return SELVAL;
}


/* CCCCCCCCC  CROSSOVER  CCCCCCCCCCCCC */
int crossover (SELPAR1,SELPAR2)
int SELPAR1;
int SELPAR2;

{
int j=0;
int JCROSS = 0;
c1=&CHILD1;
```

```
c2=&CHILD2;

q= &INDIV;
q2= &INDIV2;
op= &OLDPOP;
np= &NEWPOP;

/* RRRRR Replace DOS specific random with ANSI rand(). RRR */

/* RRRR if(((float)random(100))<(PCROSS*100.00)) RRRRRRR */

if(((float)(rand() % 100)) < (PCROSS * 100.00))
    {
    JCROSS = rand() % COLS_DESIRED;
    NCROSS++;
    }
else JCROSS = COLS_DESIRED;
/* FIRST CROSSOVER -- 1 TO 1 AND 2 TO 2 */
for(j=0; j < JCROSS; j++)
    {
    *q = op->POP[SELPAR1];
    c1->CHROM[j] = mutation(q->CHROMO[j]);
    *q2 = op->POP[SELPAR2];
    c2->CHROM[j] = mutation(q2->CHROMO[j]);
    }
/* SECOND CROSSOVER -- 1 TO 2 AND 2 TO 1 */
if (JCROSS < COLS_DESIRED)
    {
    for (j=JCROSS; j < COLS_DESIRED; j++)
        {
        *q= op->POP[SELPAR2];
        c1->CHROM[j] = mutation(q->CHROMO[j]);
        *q2 = op->POP[SELPAR1];
        c2->CHROM[j] = mutation(q2->CHROMO[j]);
        }
    }
```

```
c1=&CHILD1;
c2=&CHILD2;
return JCROSS;
}
/* MMMMMMMMM  MUTATION  MMMMMMMMMMMMMMMM */

int mutation(ALLELE)
int ALLELE;
{
int MUTATE =0;
/*  IF MUTATION RATE < PMUTATION, CHANGE  1->0 OR 0->1  */
if (((float)(rand()% 100)) < (PMUTATION * 100.00))
    {
    NMUTATION++;
    if (ALLELE == 1)
        {
        MUTATE = 0;
        }
    if (ALLELE == 0)
        {
        MUTATE = 1;
        }
    }
else MUTATE = ALLELE;

/* ERROR TRAP -- IN CASE THE MUTATED BIT IS NOT AN INTEGER
1 OR 0       */
    if (MUTATE<0 || MUTATE>1)
    {
if(SHOWME) printf("\n\nYou have been hit by the big whammie!!
MUTATE:  %d\n",
MUTATE);
    }
return MUTATE;
}
```

```
/* DDDDD CHDECODE FUNCTION DDDDDDDDDDD */
float chdecode(CHILD_NO)
int CHILD_NO;
{
int i=0;
int j=COLS_DESIRED;
float X = 0.0;
float POWER_OF_2 = 1.0;

q= &INDIV;
q2= &INDIV2;

/* TURN THE CHROMOSOME 1's and 0's INTO A FLOAT VALUE FOR
'X' */

if (CHILD_NO==1) /* DECODE CHILD-NO.-ONE'S CHROMOSOME */
    {
    X=0.0;
    while (i < j)
        {
        /* if allele is a 1, increment pwr of 2 */
        if(q->CHROMO[j-1]==1)
                {
                X = X + POWER_OF_2;
                }
        POWER_OF_2 = POWER_OF_2 * 2.0;
        j--;
        }
    }
if (CHILD_NO==2)  /* else DECODE CHROMOSOME FOR CHILD NO.
TWO */
    {
    X=0.0;
    while (i < j)
        {
        /* if allele is a 1, increment pwr of 2 */
```

```
        if(q2->CHROMO[j-1]==1)
                {
                X = X + POWER_OF_2;
                }
        POWER_OF_2 = POWER_OF_2 * 2.0;
        j--;
        }
    }
return X;
}
```

```
/* GGGGGGGGGGGG GENSTATS GGGGGGGGGGGGGG */
void genstats(GEN,MAX,MIN,AVG, LIKELY_PAR1, LIKELY_PAR2)
int GEN;
float MAX;
float MIN;
float AVG;
float LIKELY_PAR1;
float LIKELY_PAR2;
{
if(SHOWME) printf("\n\t\t\t GENERATION %d IS NOW COMPLETE",
GEN);
if(SHOWME) printf("\nThe number of crossovers for this gen:\t\t%d",
NCROSS);
if(SHOWME) printf("\nThe number of mutations for this gen:\t\t%d",
NMUTATION);
if(SHOWME) printf("\nThe CUM. FITNESS (SUMFITNESS)
was:\t\t%8.5f",
SUMFITNESS);
POPMUTATIONS=POPMUTATIONS + NMUTATION;
POPCROSSES=POPCROSSES + NCROSS;
MAXMIN [(GEN-1)] [0] = MAX;
MAXMIN [(GEN-1)] [1] = MIN;
MAXMIN [(GEN-1)] [2] = AVG;
MAXMIN [(GEN-1)] [3] = LIKELY_PAR1;
MAXMIN [(GEN-1)] [4] = LIKELY_PAR2;

/* DETERMINE IF THE GEN. MAX/MIN ARE THE POP MAX/MIN
ALSO  */

if (MAX > POPMAX)
    {
    POPMAX = MAX;
    POPMAXGEN = GEN;
    BEST_POP_PAR1 = LIKELY_PAR1;
    BEST_POP_PAR2 = LIKELY_PAR2;
```

```
    }

if (MIN < POPMIN)
    {
    POPMIN = MIN;
    POPMINGEN = GEN;
    }
/* RESET GENERATION VARIABLES AND COUNTERS*/
MAX=0.0;
MIN=0.0;
AVG=0.0;
LIKELY_PAR1 = 0.0;
LIKELY_PAR2 = 0.0;
NMUTATION=0;
NCROSS=0;
}
```

```
/* PPPPPPPPPPPP   POPSTATS   PPPPPPPPPPPP */

void popstats()
{

int l=0; /* counter for printing the rows of MAX, MIN, and AVG history */
int m=0; /* counter for printing the cols. of MAX, MIN, AND AVG */
int POP_MAX_GEN = 0; /* the generation in which the pop. max. occurs. */
float VALUE; /* temp variable to assist in printing array */
float POPULATION_MAX=0.0;
float GUESS_RATIO = 0.0;
if(SHOWME) printf("\n\nThe Maximum Fitness for the Population was:
%10.5f\n",
POPMAX);
if(SHOWME) printf("\n\t\t\tand occurred in Generation  %d\n",
POPMAXGEN);
if(SHOWME) printf("\nThe Minimum Fitness for the Population was:
%10.5f",
POPMIN);
if(SHOWME) printf("\n\t\t\tand occurred in generation  %d\n",
POPMINGEN);
if(SHOWME) printf("\n\n\tThe MAX, MIN, AND AVG FOR EACH
GENERATION
FOLLOWS:");

if(SHOWME) printf("\nGEN");
printf("\n\t\tMAX\t\tMIN\t\tAVG\tLIKELY PARENT1\tLIKELY
PARENT2");
   for (l=0; l<MAXGEN; l++)
   {
   if(MAXMIN [l] [0] > POPULATION_MAX)
      {
      POPULATION_MAX = MAXMIN[l] [0];
      POP_MAX_GEN = l+1;
      }
   printf("\n");
```

```
    if(SHOWME) printf("%d", l+1);
        for (m=0; m<5; m++)
                {
                VALUE = MAXMIN [l] [m];
                printf("\t%10.5f", VALUE);
                }
    }
printf("\n\n The Population Max was  %8.5f and occurred in gen.  %d.\n",
    POPULATION_MAX, POP_MAX_GEN);
printf("\n Therefore, my best rough guess of the source integers is:");
printf("\n %5.2f and %5.2f", BEST_POP_PAR1, BEST_POP_PAR2);
GUESS_RATIO = BEST_POP_PAR1 / BEST_POP_PAR2;
printf("\n And the ratio is: %9.5f ", GUESS_RATIO);
printf("\n Reminder: your target value was:  %9.5f ", TARGET);
}
```

# APPENDIX B

## PERFORMANCE OF THE INTEGER2 GENETIC ALGORITHM

The 2000-generation runs of INTEGER2.EXE depicted in this appendix were each completed in roughly 26 seconds on an Intel 80486 / 33 Mhz, IBM compatible computer running MSDOS 5.0 and operating in the Windows environment. The size of the population (MAXPOP) was set to 100 individuals, and the number of generations (MAXGEN) was set to 2000. The chromosome length was eight.

Experimentation with the Genetic Algorithm supported the literature in that some degree of mutation was necessary to provide diversity within the population, but there is a limit at which the mutation rate becomes destructive to the optimization search. We found that we achieved the best performance when we set the probability of mutation to be 3%. The other values we looked at were 8%, 10%, and 50%, respectively, but our best results came from using a 3% mutation rate.

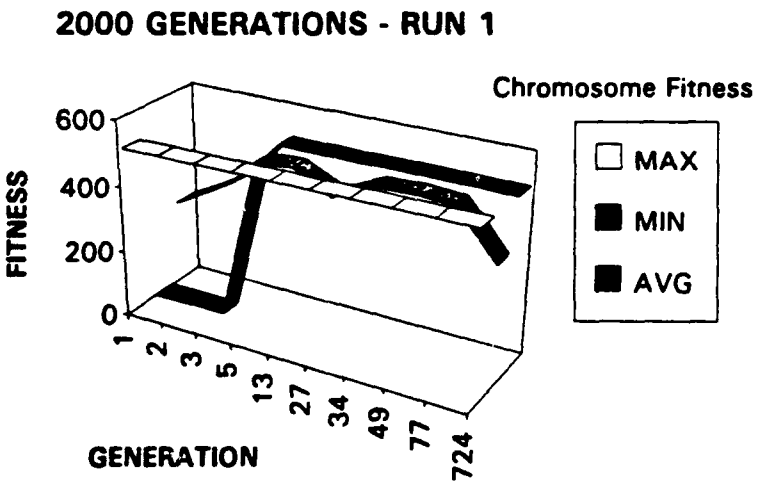
The first five of the following diagrams represent runs with no crossover and 3% mutation rate (runs 1 through 4). The next four are sample runs with probability of crossover of 50% and mutation rate of 3%. The first diagram shows the text output of INTEGER1 when the user enters a "0" when prompted for type of output desired.

The following graphs represent the value of having crossover. Our best results came with 50% probability of crossover. (Note: the only generations graphed are those in which the max fitness changed -- other X axis points were omitted for clarity of the graphs).

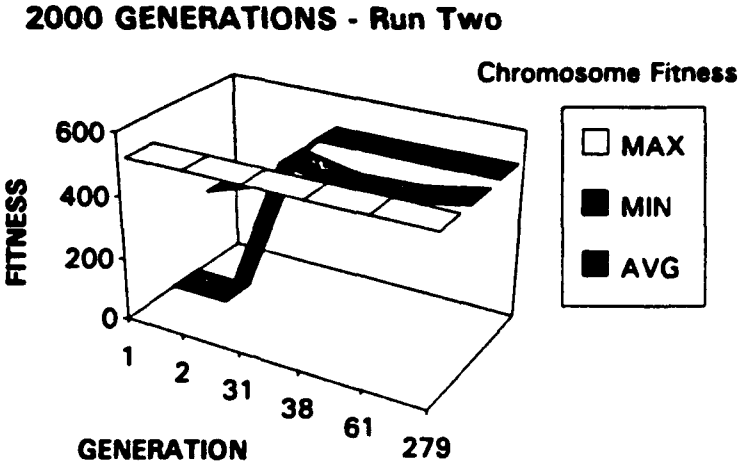
This is the first run output (subsequent outputs are shown as graphs).

| RUN ONE | | Probability of Crossover = 0.0 | | | |
|---|---|---|---|---|---|
| Please enter the probability of mutation (float value only): | | | | | |
| Your selected value was: 0.30 | | | | | |
| Enter '1' to view chromosomes and stats. '0' for summary only. | | | | | |
| Enter the target float value: | | | | | |
| GEN | MAX | MIN | AVG | | |
| 1 | 509.4912 | 1 | 253.8391 | 132 | 46 |
| 2 | 510.6805 | 1 | 315.9073 | 207 | 51 |
| 3 | 511.3784 | 1 | 386.2252 | 18 | 3 |
| 5 | 511.83 | 498.6641 | 508.4904 | 250 | 48 |
| 13 | 511.9133 | 506.6862 | 509.1095 | 127 | 24 |
| 27 | 511.9216 | 444.8784 | 506.7358 | 53 | 10 |
| 34 | 511.9498 | 501.7355 | 508.6539 | 76 | 14 |
| 49 | 511.9645 | 505.8784 | 508.5961 | 187 | 35 |
| 77 | 511.9966 | 497.7117 | 507.9379 | 215 | 40 |
| 724 | 511.9991 | 361.3784 | 505.8761 | 156 | 29 |
| | | | | | |
| The Population Max was 511.99905 and occurred in gen. 724. | | | | | |
| | | | | | |
| Therefore, my best rough guess of the source integers is: | | | | | |
| 156.00 and 29.00 | | | | | |
| And the ratio is: 5.37931 | | | | | |
| Reminder: your target value was: 5.37837 | | | | | |

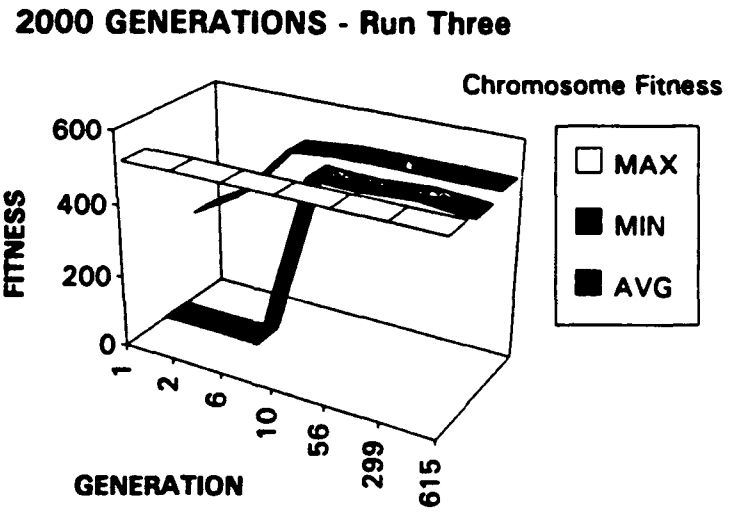
This is the chart for the first run, probability of mutation is 3%, no crossover.



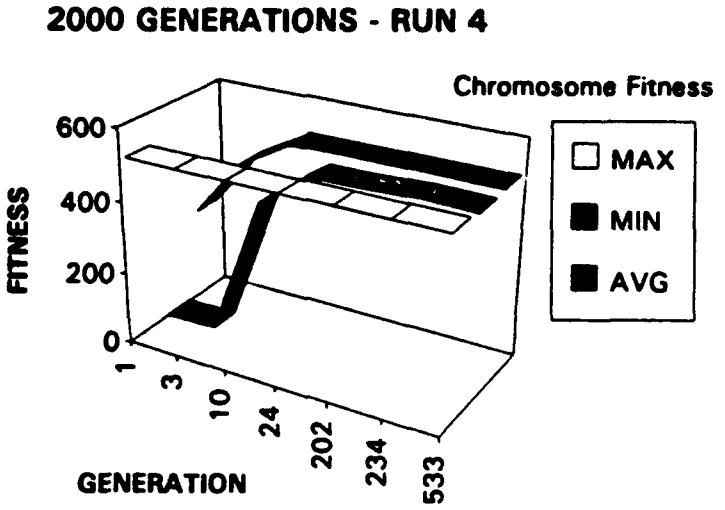
This is the diagram for the second run, pmutation is 3%, no crossover.



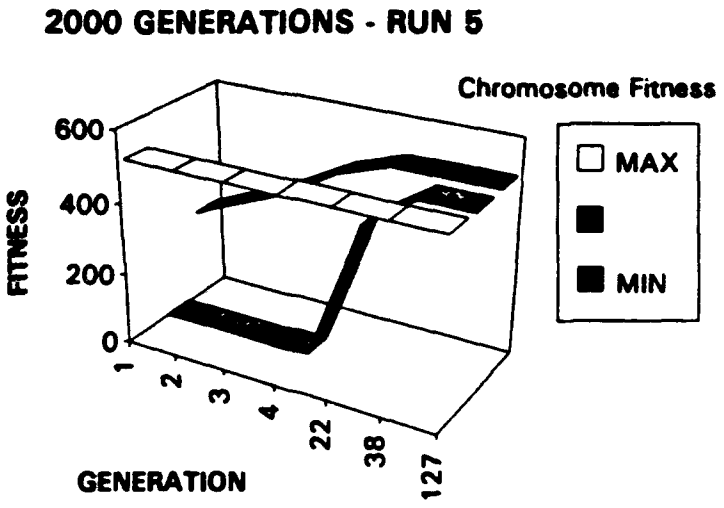

This is the chart for the third run, pmutation is 3%, no crossover.



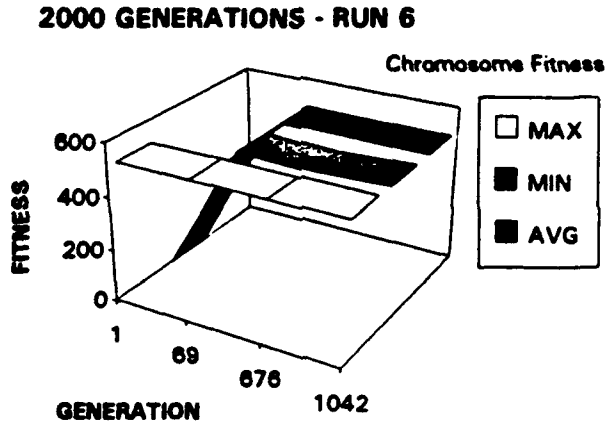
This is the chart for the fourth run, pmutation is 3%, no crossover.



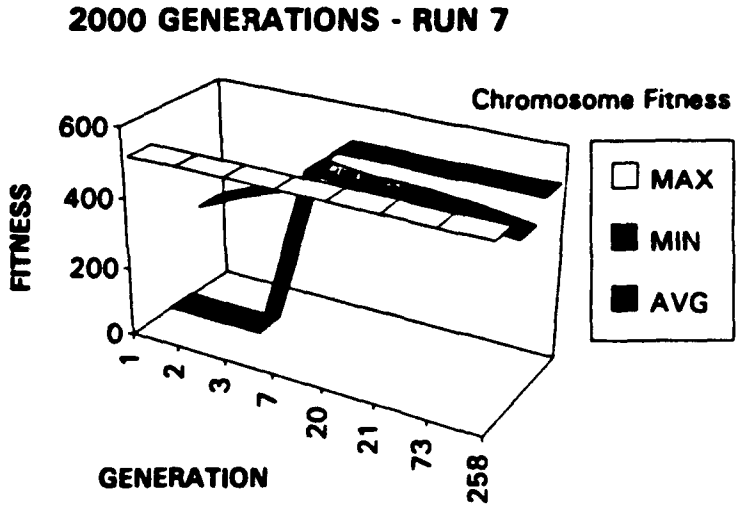
This is the chart for the first run with new probability of crossover of 0.5.



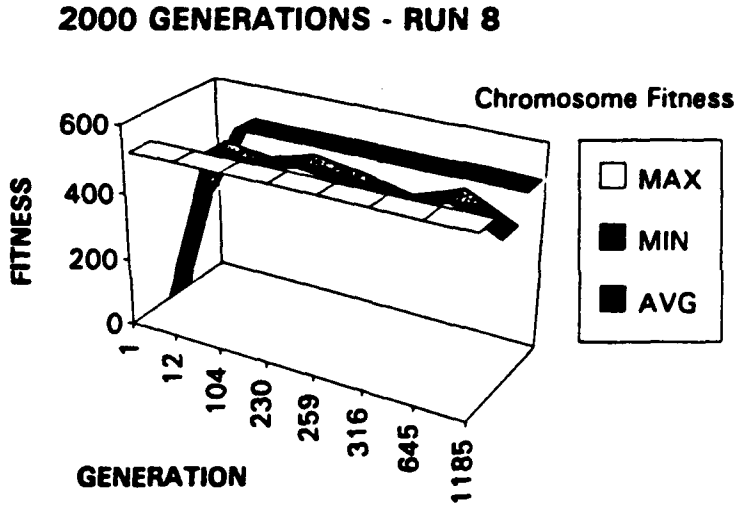

This is the diagram for the second run with new probability of cross of .5



This is the chart for the third run of probability of crossover of .5.




This is the fourth run of new probability of crossover of .5.



## LIST OF REFERENCES


Anderson, David Ray, Dennis J. Sweeny, and Thomas Arthur Williams, An Introduction to Management Science: Quantitative Approaches to Decision Making, West Publishing Co., St. Paul Minnesota, 1991.

Bhargava, Hemant K., and Michael E. Culpepper, *"Combat Forces as Autonomous Agents in Combat Simulators: An Overview."*, 1992.

Branley, William C., *"Modeling Observation In Intelligent Agents: Knowledge And Belief,"* M.S. Thesis, Naval Postgraduate School, Monterey, California, March, 1992.

Braudaway, Wesley, *"A Blackboard Approach To Computer Generated Forces,"* Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation, Orlando, FL, March, 1993.

Culpepper, Michael E., *"Tactical Decision Making in Intelligent Agents: Developing Autonomous Forces in NPSNET,"* M.S. Thesis, Naval Postgraduate School, Monterey, California, March, 1992.

Darwin, Charles L., On the Origin of Species by Means of Natural Selection, 1859.

Davis, Lawrence, Handbook of Genetic Algorithms, Van Nostrand Reinhold, New York, NY, 1991.

Forsyth, R., Expert Systems Principles and Case Studies, Second Edition, Chapman and Hall Computing, New York, NY, 1989.

Gat, Erann, et al., *"Semi-Automated Forces for Corps Battle Simulation,"* Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation, Orlando, FL, March, 1993.

Goldberg, David E., Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley Publishing Co., Reading, Mass, 1989.

Grefenstette, John J., and Connie L. Ramsey, *"An Approach to Anytime Learning,"* Proceedings of the Ninth International Machine Learning Conference, Aberdeen Scotland, 1992.

Henderson, Thomas C., Patrick Dalton, and Joseph L. Zachary, *"ℜ: A Research Program for Autonomous Agent Behavior Specification and Analysis,"* Proceedings of the 1991 IEEE International Symposium on Intelligent Control, IEEE Inc., New York, 1991.

Holland, John H., Adaptation in Natural and Artificial Systems, MIT Press, Cambridge, Massachusetts, 1992.

Johnson, R. Colin, *"What Is Cognitive Computing?"* Dr. Dobb's Journal, p. 18, February, 1993.

Koza, J.R., Genetic Programming, MIT Press, Cambridge, Massachusetts, 1992.

Kwak, S. H., R. B. McGhee, and T. E. Bihari, *"Rational Behavior Model: A Tri-level Multiple Paradigm Architecture For Robot Vehicle Control Software,"* Naval Postgraduate School, March, 1992.

Maigret, Pascal, *"Experiments in Reactive Planning and Control with Mobile Robots,"* Proceedings of the 1991 IEEE International Symposium on Intelligent Control, IEEE Inc., New York, 1991.

Page-Jones, Meilir, The Practical Guide to Stuctured Systems Design, Prentice-Hall Inc., Englewood Cliffs, N.J., 1988.

Paquet, E. B. Chaib-draa, and L. Lamontagne, *"Integrating Reaction, Planning and Deliberation in Architecture for Multiagent Environment,"* Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation, Orlando, FL, March, 1993.

Pratt, David R., and others, *"NPSNET: A Networked Vehicle Simulation With Hierarchical Data Structures,"* Naval Postgraduate School, 1992.

Reddy, Robert, *"ASTO Program Review,"* Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation, Orlando, FL, March, 1993.

Schultz, A.C., *"Using a Genetic Algorithm to learn Strategies for Collision Avoidance and Local Navigation,"* AIC-91-018, Navy Center for Applied Research in Artificial Intelligence, Naval Research Laboratory, Washington, D.C., 1991.

Schultz, A.C., and J.J. Grefenstette, *"Using a Genetic Algorithm to Learn Behaviors for Autonomous Vehicles,"* AIC-92-009, Navy Center for Applied Research in Artificial Intelligence, Naval Research Laboratory, Washington, D.C., 1992.

Siksik, D. N., *"Intelligent Computer Generated Forces Through Expert Systems,"* Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation, Orlando, FL, March, 1993.

Winston, Patrick H., Artificial Intelligence, Addison-Wesley, Reading, MA, 1984.

## INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center — 2
   Cameron Station
   Alexandria, VA 22304-6145

2. Library, Code 0142 — 2
   Naval Postgraduate School
   Monterey, CA 93943-5002

3. Dr. David Pratt — 1
   Naval Postgraduate School
   Code CS/PD, Computer Science Dept.
   Monterey, CA 93943-5100

4. Institute For Defense Analysis — 1
   Simulation Laboratory
   1801 N. Beauregard St.
   Alexandria, VA 22311

5. Professor Hemant Bhargava — 2
   Naval Postgraduate School
   Code AS/BH, Administrative Science Dept.
   Monterey, CA 93943-5000

6. Professor B. Ramesh — 1
   Naval Postgraduate School
   Code AS/BR, Administrative Science Dept.
   Monterey, CA 93943-5000

7. CAPT Robert Jacobs — 2
   8548 S.W. 114th Place
   Miami, FL 33173

8. Lt John Steiner — 2
   c/o 5757 S. Staples
   Apt# 2319
   Corpus Christi, TX 78413

9. Professor Carl Jones — 1
   Naval Postgraduate School
   Code AS/CJ, Administrative Science Dept.
   Monterey, CA 93943-5000

10. David L. Neyland — 1
    Program Manager
    ARPA-ASTO
    3701Fairfax Dr.
    Arlington, VA 22203-1714

11. Dr. Michael Zyda — 2
    Naval Postgraduate School
    Code CS/ZM, Computer Science Dept.
    Monterey, CA 93943-5100

12. Director, Training and Education — 1
    MCCDC, Code C46
    1019 Elliot Road
    Quantico, VA 22134-5027